ARL-TR-7363 ● AUG 2015

US Army Research Laboratory

Data Analysis Tools for Visualization Study

by Richard L Astrom

Approved for public release; distribution is unlimited.

NOTICES

Disclaimers

ARL-TR-7363 ● AUG 2015

US Army Research Laboratory

Data Analysis Tools for Visualization Study

by Richard L Astrom
Computational and Information Sciences Directorate, ARL

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. REPORT DATE *(DD-MM-YYYY)* Aug 2015	**2. REPORT TYPE** Final	**3. DATES COVERED (From - To)** October 1, 2014 – December 31, 2014
4. TITLE AND SUBTITLE Data Analysis Tools for Visualization Study		**5a. CONTRACT NUMBER** W911QX-07-F-0023 **5b. GRANT NUMBER** **5c. PROGRAM ELEMENT NUMBER**
6. AUTHOR(S) Richard L Astrom		**5d. PROJECT NUMBER** **5e. TASK NUMBER** **5f. WORK UNIT NUMBER**
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) ICFI, Cybersecurity & Systems Division 7125 Thomas Edison Drive, Suite 100 Columbia, MD 21046		**8. PERFORMING ORGANIZATION REPORT NUMBER** ARL-TR-7363
9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES) US Army Research Laboratory ATTN: RDRL-CIN-D 2800 Powder Mill Road Adelphi, MD 20783-1138		**10. SPONSOR/MONITOR'S ACRONYM(S)** ARL **11. SPONSOR/MONITOR'S REPORT NUMBER(S)**

12. DISTRIBUTION/AVAILABILITY STATEMENT

Approved for public release; distribution is unlimited.

13. SUPPLEMENTARY NOTES

14. ABSTRACT

A study in fiscal year (FY) 2014 led by Dr Robert Erbacher, described in *Evaluation of the Presentation of Network Data via Visualization Tools for Network Analysts*, resulted in a large body of data from trial results. This report presents tools that I have developed to codify, display, and analyze the data. In the current visualization study, test subjects looked at a set of network intrusion alerts and decided which of those alerts represented true threats. The correct answers and the selections by each subject were recorded as fixed-format text files. My tools parse this text, insert the data into tables in a relational database, and create views to facilitate reading selected data from the tables. Additional tables were built, which contained sums of the performance statistics for each trial. I wrote Python programs to analyze the summary data and applied statistical tests to some of the means to determine whether they were statistically different. A plot capability was added as well.

15. SUBJECT TERMS

Visualization, alerts, network analysis, true positives, false positives, true negatives, false negatives

16. SECURITY CLASSIFICATION OF:			**17. LIMITATION OF ABSTRACT**	**18. NUMBER OF PAGES**	**19a. NAME OF RESPONSIBLE PERSON**
a. REPORT	**b. ABSTRACT**	**c. THIS PAGE**			Richard L Astrom
Unclassified	Unclassified	Unclassified	UU	60	**19b. TELEPHONE NUMBER (Include area code)** 301-394-3181

Standard Form 298 (Rev. 8/98)
Prescribed by ANSI Std. Z39.18

Contents

List of Figures v

List of Tables v

1. Introduction 1

2. Methods 1

2.1 Test Subjects 1

2.2 Alerts Presented 1

2.3 Three Display Types 2

2.4 Inputs from Test Subjects 3

3. Subject Trial Results 4

3.1 Selection Text Files 4

3.2 Creation of Table subject_choices 4

4. Summaries of Trial Data 5

4.1 True Answers 5

4.2 SQL Join Between Subject Choices and True Answers 6

4.3 Statistics for Each Trial 6

4.4 Results Summary Table 6

5. Correlations and Plots 7

5.1 Mean *t* Tests 7

5.2 Mean *F* Tests 8

5.3 Scatter Plot Example 8

6. Future Analysis 10

7. References and Notes 11

Appendix A. SQL Table Generation 13

Appendix B. Python Programs **25**

Appendix C. Results Summary Listing **45**

List of Symbols, Abbreviations, and Acronym **51**

Distribution List **52**

List of Figures

Fig. 1 Node-link: The user is asked here to determine regions of the visualization that imply intrusions and intrusion attempts by clicking near a particular link or node (from Etoty et al.[1]) 2

Fig. 2 Table: The user is asked here to determine which alert messages in the table imply intrusions and intrusion attempts by clicking the checkboxes in the Suspicious column (from Etoty et al.[1]) 3

Fig. 3 PC: The user is asked here to determine which alert messages in the table imply intrusions and intrusion attempts by clicking on suspicious links (from Etoty et al.[1])..3

Fig. 4 ARL: F1 score for the first, second, and third trial...............................9

Fig. 5 MSU: F1 score for the first, second, and third trial 9

List of Tables

Table A-1 Truth table - scores..19

Table A-2 SQL Table *means_compare*..22

INTENTIONALLY LEFT BLANK.

1. Introduction

A visualization study in fiscal year (FY) 2014, led by Dr Robert Erbacher,[1] resulted in a large body of data from trial results. To process the results, I have developed tools to codify, display, and analyze the data.

In the visualization study, test subjects looked at a set of network intrusion alerts and decided which of those alerts represented true threats. The set of test alerts was presented to 51 test subjects, each of whom tried the same task with 3 different display types presented to them. This procedure resulted in 145 separate tests, because some of the test subjects did not complete all 3 tasks. The test subject's objectives were to 1) identify and mark all of the alerts that were true threats and 2) avoid marking any that were not. The correct answers and the selections by each subject were recorded as fixed-format text files.

My tools parse the text files and insert the data into tables in a structured query language (SQL) relational database. I used PostgreSQL as the SQL application, called from Python programs running on a Linux Red Hat operating system. I created views to facilitate reading selected data from the tables, as well as built additional tables, which contained sums of the performance statistics for each trial.

Next I wrote Python programs to analyze the summary data, and applied statistical tests to some of the means to determine whether they were statistically different. A plot capability was added as well.

This report describes all of these tools and the development process used to create them.

2. Methods

2.1 Test Subjects

The 51 test subjects came from 2 distinct groups. The first group consisted of US Army Research Laboratory (ARL) network analysts who were experienced at doing this task. The second group consisted of students at Morgan State University (MSU) who had no experience. These 2 groups are identified in the SQL tables by their organization as ARL or MSU.

2.2 Alerts Presented

The subjects were presented with 140 alerts, 42 of which represented real threats. A perfect score by a subject constituted selection of all 42 threats, with no additional alerts selected. A senior network analyst designed the inputs, which were designed so they would be accurate and representative of the problem.

2.3 Three Display Types

The subjects did the same task using 3 different display formats: node-link, table, and parallel coordinate (PC). Figures 1, 2, and 3 illustrate the 3 types of displays (which are reproduced with permission from Etoty et al.[1]).

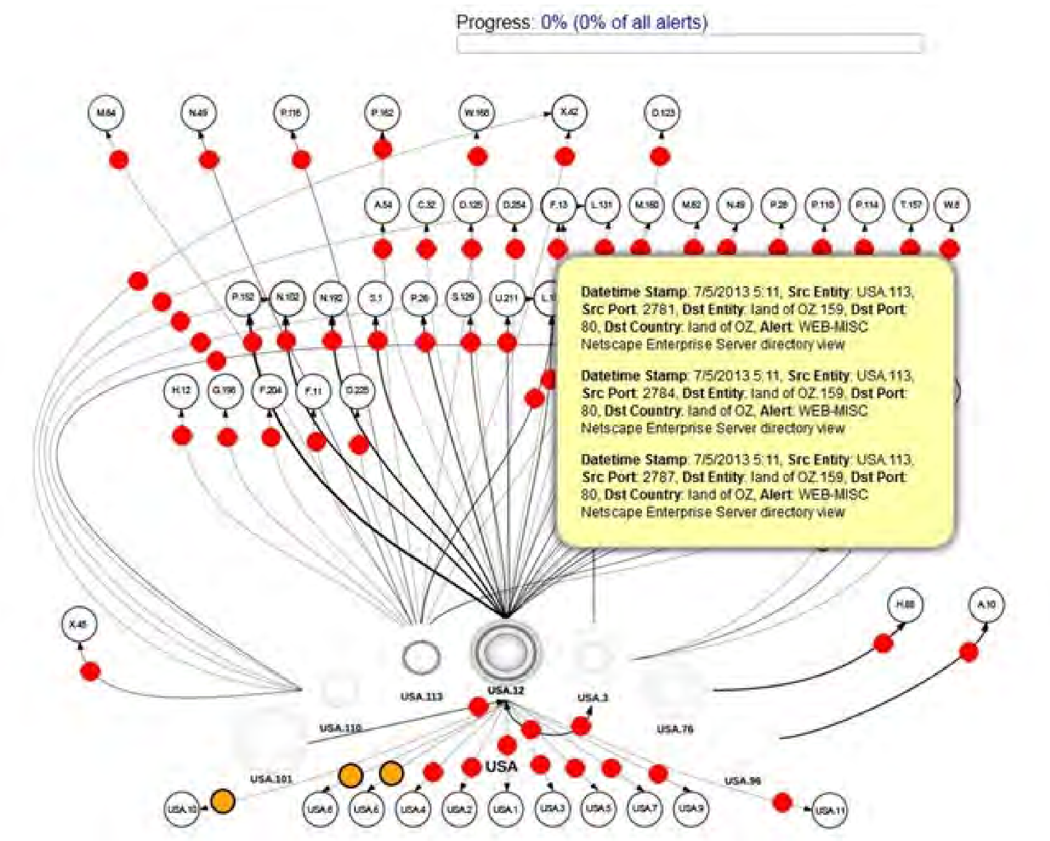
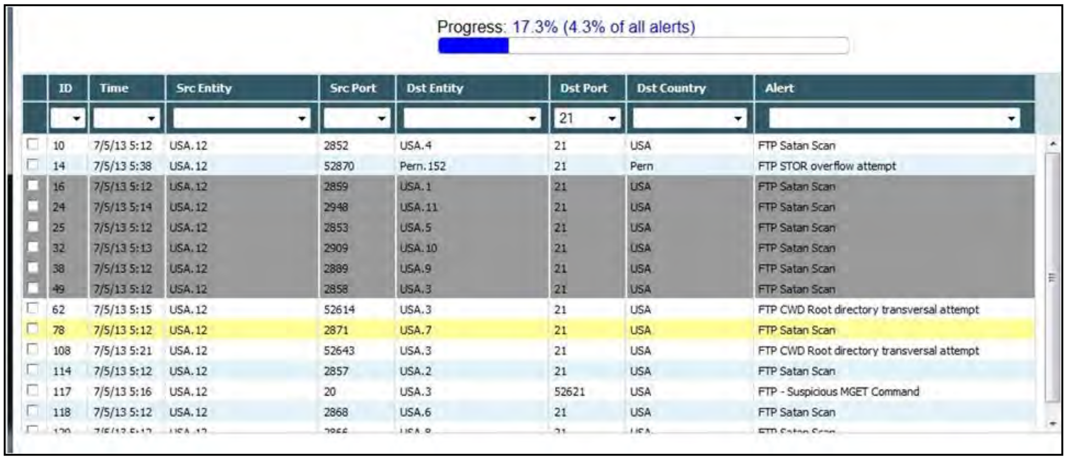


Fig. 1 Node-link: The user is asked here to determine regions of the visualization that imply intrusions and intrusion attempts by clicking near a particular link or node (from Etoty et al.[1])



Progress: 17.3% (4.3% of all alerts)

	ID	Time	Src Entity	Src Port	Dst Entity	Dst Port	Dst Country	Alert
						21		
☐	10	7/5/13 5:12	USA.12	2852	USA.4	21	USA	FTP Satan Scan
☐	14	7/5/13 5:38	USA.12	52870	Pern.152	21	Pern	FTP STOR overflow attempt
☐	16	7/5/13 5:12	USA.12	2859	USA.1	21	USA	FTP Satan Scan
☐	24	7/5/13 5:14	USA.12	2948	USA.11	21	USA	FTP Satan Scan
☐	25	7/5/13 5:12	USA.12	2853	USA.5	21	USA	FTP Satan Scan
☐	32	7/5/13 5:13	USA.12	2909	USA.10	21	USA	FTP Satan Scan
☐	38	7/5/13 5:12	USA.12	2889	USA.9	21	USA	FTP Satan Scan
☐	49	7/5/13 5:12	USA.12	2858	USA.3	21	USA	FTP Satan Scan
☐	62	7/5/13 5:15	USA.12	52614	USA.3	21	USA	FTP CWD Root directory transversal attempt
☐	78	7/5/13 5:12	USA.12	2871	USA.7	21	USA	FTP Satan Scan
☐	108	7/5/13 5:21	USA.12	52643	USA.3	21	USA	FTP CWD Root directory transversal attempt
☐	114	7/5/13 5:12	USA.12	2857	USA.2	21	USA	FTP Satan Scan
☐	117	7/5/13 5:16	USA.12	20	USA.3	52621	USA	FTP - Suspicious MGET Command
☐	118	7/5/13 5:12	USA.12	2868	USA.6	21	USA	FTP Satan Scan

Fig. 2 Table: The user is asked here to determine which alert messages in the table imply intrusions and intrusion attempts by clicking the checkboxes in the Suspicious column (from Etoty et al.[1])

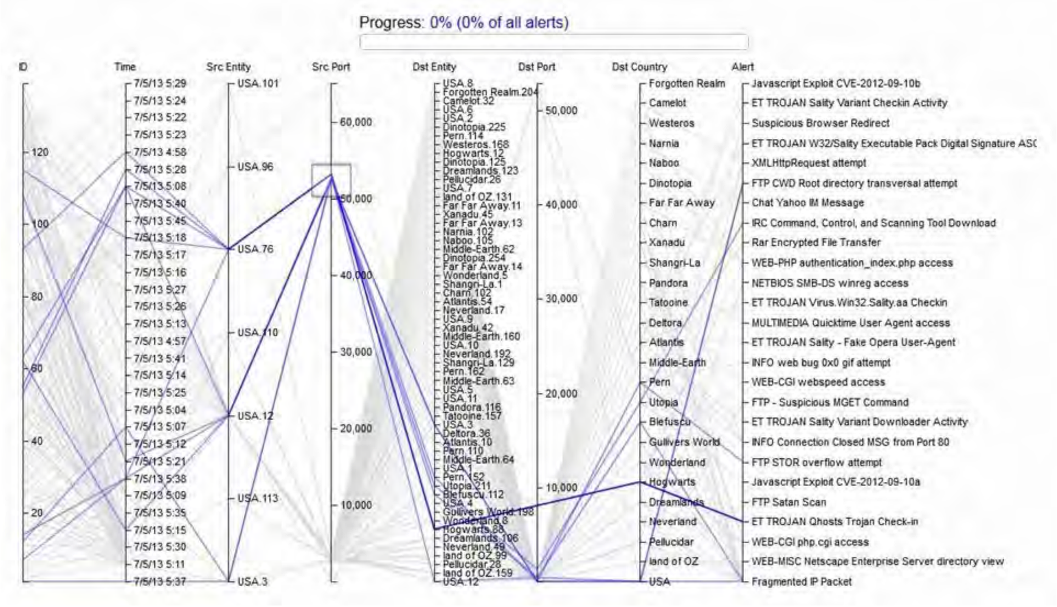


Fig. 3 PC: The user is asked here to determine which alert messages in the table imply intrusions and intrusion attempts by clicking on suspicious links (from Etoty et al.[1])

2.4 Inputs from Test Subjects

Each subject, on each of the 3 display types, was asked to select which alerts they thought were true threats and provide comments, which could be typed within the line. Their comments were sorted by group for multiple alerts that the subject thought were related to each other. The results of each trial were recorded in a text file.

Overall, 145 text files, showing each subject's selections for each trial, are included this study. The report describes the SQL database I built, which contains all of the information and how analysis structures and programs were built to analyze the data. There are sections describing the following:

- SQL table design and creation
- Parsing of the results text files
- Creation of views into the SQL tables, showing summaries of the data
- Correlations and data plots of the summary data
- Future plans, including additional correlations and plots
- SQL table generation details (Appendix A)
- Listings of all the Python programs (Appendix B)
- Listing of the results summary for the 145 trials (Appendix C)

3. Subject Trial Results

3.1 Selection Text Files

For each subject and each display type, there is a text file showing the subject's selections of suspicious alerts. A sample text file is shown at the beginning of Appendix A. These were parsed by a Python program, *process_main.py*, which is described in Appendix A and listed in Appendix B.

3.2 Creation of Table subject_choices

The parsing output was inserted into the SQL table *subject_choices*.[2] This table has 1 row for every subject/display/selection. For example, if Subject 16, using the node-link display, selects 42 of the 140 alerts presented, then that adds 42 lines to *subject_choices*.

Each row of *subject_choices* contains the following:

- The subject's organization
- Subject identification (ID)
- Display type
- Start time

- Alert number selected
- Alert group for this selection
- Alert group comment
- Order of this display type for this subject

The subjects averaged approximately 50 selections for each trial, with a total of 7,227 subject selections. This is the number of rows in the table *subject_choices*.

The detailed description of this process is in Appendix A. The Python programs are listed in Appendix B.

4. Summaries of Trial Data

With this large number of trial selections by each subject, a summary for each subject/display combination was needed. A very simple SQL query gives the number selected in each trial. However, what is really needed is a sum of all decisions, separated into 4 categories:

- True positive (TP), those that were correctly selected: GOAL = 42
- True negative (TN), those that were correctly NOT selected: GOAL = 98
- False positive (FP), those that were selected in error: GOAL = 0
- False negative (FN), those that were not selected but should have been selected: GOAL = 0

The sum of all 4 of these categories will always be 140, the number of alerts presented in the trials. The sum of TP and FN will always be 42, the number of actual threats among the 140 total alerts. The sum of TN and FP will always be 98, the number of non-threat alerts.

4.1 True Answers

To sort the subjects' decisions into these 4 categories, it was necessary to have the right answers in a SQL table. That table, *true_alerts*, was built with 140 rows, 1 for each alert, along with the correct answer for that alert. I also constructed a view called *true_threats* in *true_alerts*, showing only the threats. Example listings of these tables are shown in Appendix A.

One more piece of information about the true threats was provided—whether the test set designer considered them easy, moderate, or hard to identify. So, for each alert identified as a threat, there are 3 extra columns, identifying each threat as easy, moderate, or hard. There are 5 easy, 5 moderate, and 32 hard alerts.

4.2 SQL Join Between Subject Choices and True Answers

With the tables *subject_choices* and *true_alerts* in place, a Python program prepared a SQL JOIN between these 2 tables, correlating each selection with its true answer and filling in the fields TP, FP, TN, and FN. These were then summed to create one SQL table row for each subject/display, for 145 rows. These were inserted into the results table *results_summary_plus*.

4.3 Statistics for Each Trial

Three more statistics were calculated for each trial:

1) Recall is the percentage of real threats that were correctly identified.
    - Recall = TP/(TP + FN)

2) Precision is the percentage of selections actually representing real threats (also called the false alarm rate [FAR]).
    - Precision = TP/(TP + FP)

3) The F1 score represents a geometric average of Recall and Precision.
    - F1 = 2*TP/(2*TP + FP + FN)

All of these scores have a maximum of 1, which is the best score. For the subdivision of the true alerts into easy, moderate, and hard, the sample sizes were too small for Precision and F1 to be meaningful; therefore, only the Recall score was calculated for the 3 subcategories. Recall shows the percentage of true alerts in each subcategory that were correctly selected.

4.4 Results Summary Table

With all of these fields to calculate, I created a summary table with the basic data first and then a second final table based on the first. This explains the name of the resulting SQL table, *results_summary_plus*. The Python programs and SQL commands used to create this table are in Appendix A, and the actual Python listings are in Appendix B.

The *results_summary_plus* table, with 145 rows, has the following fields:

- Basic Trial information: *org, subject, display, time_order, completion_time*

- Overall Scores: *tp, fp, tn, fn, recall, precision, f1_score*

- Subcategory Scores: *tp_easy, fn_easy, recall_easy, tp_mod, fn_ mod, recall_ mod, tp_hard, fn_ hard, recall_ hard*

A SQL View was also created into *results_summary_plus*, called *seven_scores*. This View shows summaries useful for analysis. It reduces the display to the following columns: *org, subject, display, completion_time, tp, fp, tn, fn, recall, precision, f1_score*.

This summary is listed for all 145 trials in Appendix C.

5. Correlations and Plots

Comparisons of different groups of trial results are easily done using SQL queries on the table *results_summary_plus*. Python programs were written to apply both *t* tests and *F* tests to the means of groups of data. They use the Python statistical package *scipy.stats*.[3] The student's *t* test compares the means of 2 distributions. The *F* test compares multiple means, 2 or more.

The Python statistical library was installed, and both *t* tests and *F* tests were used and tested. The Python plotting package supplied by MATLAB, *matplotlib*, was also installed and used.

5.1 Mean *t* Tests

The student's *t* test compares 2 means from different distributions.[4] The outcome of a *t* test is a determination if the 2 means are statistically different. For a first demonstration of *t* tests, 10 scores from each of the 3 display types were tested, for a total of 30 tests. They were tested to see whether the performance was statistically different between the ARL and MSU subjects.[5]

For the purpose of documenting the first set of mean comparisons, I designed a SQL table to hold the results. The function *compare_means_t*, called by the Python main program *means_test_main.py*, inserts results into the table as it calculates them. This table, *means_compare*, is listed in Appendix A. It has 30 different score comparisons.

As an example, looking at the F1 score, the ARL analysts were better at 95% confidence using the table display, which is expected because that is similar to the display they use every day. However, using the other 2 displays, they are not better to a 95% confidence level.

5.2 Mean *F* Tests

The *F* test compares multiple means from different distributions, any number 2 or more.[6] The outcome of an *F* test is a determination of whether any of the means are statistically different from the others. It does not tell you which one or ones. A Python method, *compare_means_f*, was written to perform this test.

An example from the trial data was run, comparing the F1 scores for each subject's first, second, and third trial. (The display order was random, so each subject's order of displays is different, with 6 different combinations.) The Python log is shown in Appendix A.

Neither the ARL group nor the MSU group showed significant change for the second and third times through the trial. These datasets are used again in the next example, in a scatter plot.

5.3 Scatter Plot Example

One plot has been created, as an example of the plot capability available for future analysis: a scatter plot of all F1 scores for the first, second, and third trial for each subject. The means and their 95% confidence intervals are superimposed on the scatter chart, along with a trend-line plot. Neither the ARL group nor the MSU group showed an improvement for the second and third times through the trial.

Figures 4 and 5 show the 2 plots. There are many more types of plots, and different types will be used in future analysis.

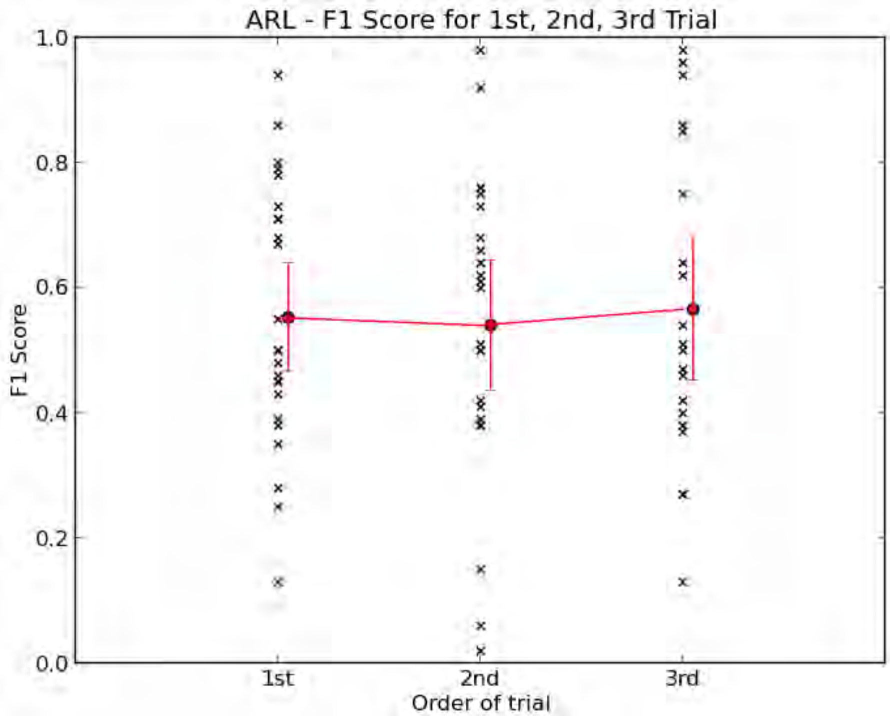


Fig. 4 ARL: F1 score for the first, second, and third trial

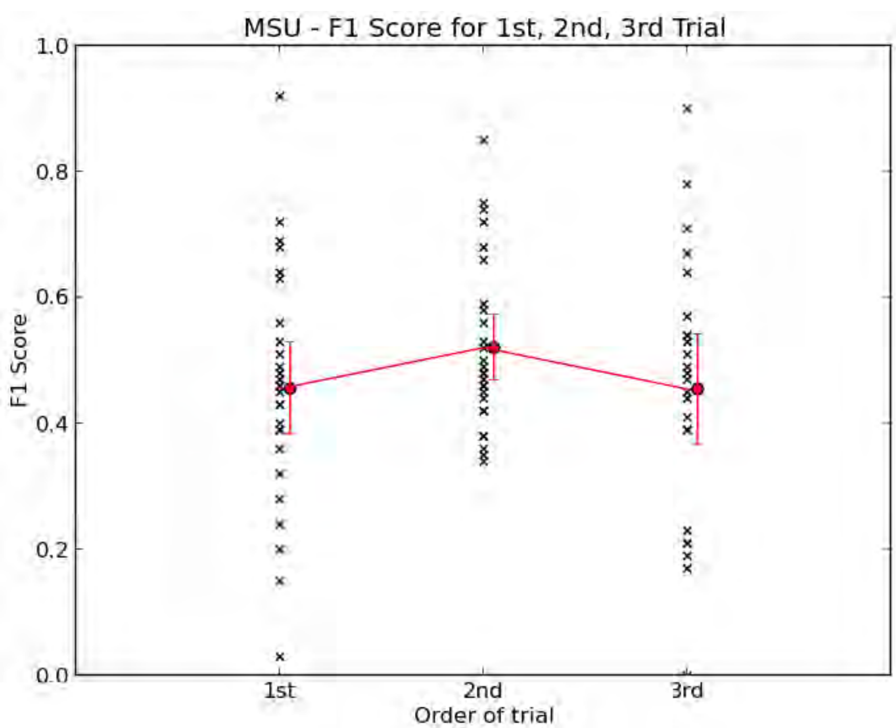


Fig. 5 MSU: F1 score for the first, second, and third trial

6. Future Analysis

The visualization study included an extensive survey given to each of the 51 trial subjects, including demographic information, experience (if any) in analysis of alerts, and computer/display preferences. There are many comparisons and correlations proposed to analyze the relationship between subject performance and survey answers. More correlations and display plots are required based upon the performance in the trials. These plots will be added to the ones already done.

The following is a partial list of proposed future correlations, from the Network Science Division (NSD) project report, August 2014:[7]

- Overall performance versus experience
- Performance on event significance versus experience
- Visual preference versus performance
- Experience versus visual preference
- Performance versus age
- Visual preference versus age
- Area knowledge versus performance
- Operating system (OS), protocols, security, intrusion detection system (IDS)/intrusion prevention system (IPS), communication skills
- Area knowledge versus visual preference
- Correlation of performance against difficulty of alerts

This is not a complete list, and more analysis will be performed as needed.

7. References and Notes

1. Etoty RE, Erbacher RF, Garneau C. Evaluation of the Presentation of network data via visualization tools for network analysts. Adelphi (MD): Army Research Laboratory (US); March 2014. Report No.: ARL-TR-6865.

2. Names of Python programs and SQL tables, views, and columns are shown in italics. SQL names, by convention, are always in lowercase.

3. Scipy.org. Statistics package for Python [accessed 2014 Dec 15]. http://docs.scipy.org/doc/scipy-0.14.0/reference/tutorial/stats.html.

4. t-based confidence interval for the mean. Kalamazoo (MI): Western Michigan University; 2003 Sep 8 [accessed 2014 Dec 15]. http://www.stat.wmich.edu/s216/book/node79.html.

5. All of the examples here have been calculated using a confidence interval of 95%. This is a parameter that can be easily changed.

6. Wikipedia. F test [accessed 2014 Dec 15]. http://en.wikipedia.org/wiki/F-test.

7. Erbacher RF. *Cognitive Foundations of Cyber Analysis*, Division project report, August 2014

Bibliography

Elementary statistical calculations and simulations. Ann Arbor (MI): University of Michigan, ARC/CSCAR Python Workshop; 2013 Jun 10–14 [accessed 2014 Dec 15]. http://dept.stat.lsa.umich.edu/~kshedden/Python-Workshop/stats_calculations.html.

Etoty RE, Erbacher RF. A survey of visualization tools assessed for anomaly-based intrusion detection analysis. Adelphi (MD): Army Research Laboratory (US); April 2014. Report No.: ARL-TR-6891.

Etoty R. Network Data via visualization tools for network analysts [master's thesis]. Baltimore, MD: Morgan State University; November 2013.

Appendix A. SQL Table Generation

A-1 Creation of Table subject_choices

A-1.1 Input Text Files

The source of all the result data is the set of 145 text files, showing the alert selections for each subject using each display. The following is the text file for 1 example, Subject no. 16 from Morgan State University (MSU) on the node-link display type.

Id16-nodelink.txt:


```
{"1":[14,52,65,77,95,97,133,135,126,"The line is darker."],"2":[99,130,58,"Other
countries trying to connect directly with the
US."],"3":[57,94,105,107,109,131,30,41,47,68,72,124,"Too many
communications."],"4":[5,35,19,28,34,76,89,"Contains trojan
virus."],"5":[40,82,125,1,21,37,92,110,117,120,134,"They are suspicious
commands."]}
Time: start: 04:51:42 PM, end: 04:57:29 PM
Submitted: 04:53:03 PM; 04:54:16 PM; 04:55:40 PM; 04:56:43 PM; 04:57:18 PM;
```


Subject 16 has grouped the selections into 5 groups, "1" through "5" and has selected 42 alerts—the individual alert numbers, starting with 14,52,… For each group, Subject 16 has also written a comment (this was optional for the subjects—many are blank).

There were approximately 5 to 6 files in which changes were needed, due to something that did not match the format or was necessary to help out the parser. This included removal of quote signs (replaced by '), removal of a spurious blank line, etc. In each case, a local copy of the data was changed, not the original, and a text note describing the changes has been filed with the local copy.

A-1.2 Python Parsing Program *process_main.py*

I wrote a Python parsing program to parse this text, picking out all alerts and associating each alert with its group and comment. The parser also read the start time, for use in later determining the order of the 3 displays for this subject. This program, *process_main.py*, is listed in Appendix B-1. Parsing utility routines used by *process_main.py* are in the file named *process.py* in Appendix B-2.

The *process_main.py* file traverses the data base of subject selections, processing all 145 trials. One row in the structured query language (SQL) results table *subject_choices* is inserted for each alert selection in each trial, an average of 50 selections per trial. (The example text file used for Subject 16 contained 42 selections.)

A.1.3 Python Program to Determine Order

As part of the trial design, the order in which display types were used was different for different subjects. A second Python program, *set_order_main.py*, performs a SQL query to pick out the 3 start times for each subject, sorts them, and then inserts "1", "2', or "3" into the field *time_order* in each row of the table *subject_choices*. This program is listed in Appendix B-3.

A-1.4 Format of Table *subject_choices*

After insertion of all the subject choices into the table and the insertion of the display order for each subject, the table is complete.

The table *subject_choices* looks like the following (first few lines):

```
select * from subject_choices;
  choices_id | subject_org | subject | display  | start_time | alert_no | a_group | group_comment          |
time_order
------------+-------------+---------+----------+------------+----------+---------+-------------------------+-
-----------
      98332 | MSU         | 16      | nodelink | 16:51:42   |      135 | 1       | The line is darker. |
3
      98333 | MSU         | 16      | nodelink | 16:51:42   |      126 | 1       | The line is darker. |
3
      98334 | MSU         | 16      | nodelink | 16:51:42   |       99 | 2       | Other countries . . . |
3
      98335 | MSU         | 16      | nodelink | 16:51:42   |      130 | 2       | Other countries . . . |
3
      98336 | MSU         | 16      | nodelink | 16:51:42   |       58 | 2       | Other countries . . . |
3
      98337 | MSU         | 16      | nodelink | 16:51:42   |       57 | 3       | Too many communications.|
3
      98338 | MSU         | 16      | nodelink | 16:51:42   |       94 | 3       | Too many communications.|
3
      98339 | MSU         | 16      | nodelink | 16:51:42   |      105 | 3       | Too many communications.|
3
      . . . (7,227 rows)
```

The columns of the table are as follows:

- subject_org, subject, display, start_time, alert_no, a_group, group_comment, time_order
    - *Choices_id* is a meaningless primary key number for SQL purposes.
    - *subject_org* is US Army Research Laboratory (ARL) or MSU.
    - *subject* is the number identifying one test subject.
    - *display* is parallel coordination (PC), table, or node-link.
    - *start_time* is the starting time.
    - *alert_no* is the ID number of the alert, from 1 to 140.
    - *a_group* is the group title (usually "1," "2," etc.) assigned by the subject.
    - *group_comment* is the subject's comment about the group.
    - *time_order* is the order in which this subject used this display—first, second, or third.

A-2 Creation of Table results_summary_plus

A-2.1 Need for a Results Summary

For each subject and each display, the results summary was needed to show how many alerts the subject chose correctly (true positive [TP]), how many were chosen incorrectly (false positive [FP]), how many real threats were not chosen (false negative [FN]), and how many nonthreats were not chosen (true negative [TN]). Higher TPs and TNs represent better performance. For each trial, these sums constitute the basic information needed for further research and analysis of the trials. All of the further work in this report and future reports will have the performance summations as a basis.

A-2.2 Correct Answers: *true_alerts* Table and *true_threats* View

To correctly identify each of these values for a particular selection or lack thereof, the correct answers were needed in a table. These correct answers were already available in an Excel spreadsheet and were imported into a SQL table. The table *true_alerts* shows the correct answer for all 140 alerts. If the alert is not threatening, all fields are blank. If the field is a threat, the field *true_alert* is set to 1. In addition, the true threats were designated as *easy*, *moderate*, or *hard* to find. For a true threat, one of these three fields is also filled in with a 1. The following are the first few lines of *true_alerts*:

```
SELECT * FROM true_alerts LIMIT 12;
 alert_id | true_alert | moderate | hard | easy
----------+------------+----------+------+------
        1 |            |          |      |
        2 |            |          |      |
        3 |            |          |      |
        4 |            |          |      |
        5 |            |          |      |
        6 |            |          |      |
        7 |          1 |          |      |    1
        8 |            |          |      |
        9 |            |          |      |
       10 |          1 |          |    1 |
       11 |            |          |      |
       12 |            |          |      |
       . . . (140 rows)
```

A view into this table was also designed, to show only the true threats but not all alerts. That view specification and its first few lines are as follows:

```
CREATE VIEW true_threats AS SELECT * FROM true_alerts WHERE
true_alert = 1;
SELECT * FROM true_threats LIMIT 7;
 alert_id | true_alert | moderate | hard | easy
----------+------------+----------+------+------
        7 |          1 |          |      |    1
       10 |          1 |          |    1 |
       13 |          1 |          |    1 |
       14 |          1 |          |    1 |
       16 |          1 |          |    1 |
       19 |          1 |        1 |      |
       21 |          1 |          |    1 |
      . . . (42 rows)
```

A-2.3 Intermediate Table *results_summary* using JOIN

Two tables, *true_alerts* and *subject_choices*, contain the information necessary to determine the category of each selection or lack of selection by a subject in a trial. For each trial, we want to decide, for each alert ID from 1 to 140, whether it is a TP, FP, TN, or FN. After this decision is made for all 140 alerts, we need a sum of all 4 of these categories for future analysis.

The table *results_summary* contains the following fields: *org, subject, display, time_order, true_alerts, selected, tp, fp, tn, fn, f1_score, easy, tp_easy, fn_easy, mod, tp_mod, fn_mod, hard, tp_hard, and fn_hard*.

To determine which category each selection falls into, the SQL tables *subject_choices* and *true_alerts* must be combined using a JOIN command. This JOIN is performed by the Python program *create_summary_main.py*. For each trial, a SELECT statement picks all of the selections for the trial out of the table *subject_choices*, creating a temporary view called *dispnnn*, in which *disp* is the display type and *nnn* is the subject ID. (The names *dispnnn* and *dispnnnv* are the actual names used. After each trial, these temporary views are dropped, and then the names are reused for the next trial.)

```
"SELECT DISTINCT ON (subject,display) subject,
display, subject_org, time_order from subject_choices
ORDER BY subject"  # Gets all trials
"CREATE OR REPLACE TEMPORARY VIEW dispnnn AS (SELECT
subject_org,subject,time_order,alert_no,a_group,group_
comment FROM subject_choices WHERE subject = subject
AND display = display"[1]
```

[1]SQL statements are shown here without the Python mechanisms to insert variables and without extra complexity such as SQL functions COALESCE or ROUND. This process is used to make this section more readable. The actual Python code, with full SQL statement development, is in Appendix B.

Then, a second temporary view named *dispnnnv* is created by a SQL JOIN command, with a row for each of the 140 alerts. (To help understand this statement, notice that at the end, the table *true_alerts* is shortened to **t** and view *dispnnn* is shortened to **v**.) This view creation uses a JOIN, which matches rows that have the same alert ID—see the ON clause at the end. The use of "LEFT OUTER JOIN" creates a view that has a row for every row of *true_alerts*, which means that it has 1 row per alert ID, whether it is a true threat or not.

```
"CREATE OR REPLACE TEMPORARY VIEW dispnnnv AS SELECT
t.alert_id, v.subject_org AS org, v.subject,
v.time_order,
        t.true_alert AS true_alert,

        COALESCE((v.alert_no/v.alert_no),0) AS
selected,
                 # 1 if present, 0 if not present in
dispnnn

        t.true_alert*selected AS tp,
        GREATEST(selected-t.true_alert,0) AS fp,
        -GREATEST(t.true_alert,selected,0))+1 AS tn,
        GREATEST(t.true_alert-selected,0) AS fn,
        t.easy AS easy,
       t.easy*selected AS tp_easy,
        GREATEST(t.easy-selected,0) AS fn_easy,
        t.moderate AS mod,
        t.moderate*selected AS tp_mod,
        GREATEST(t.moderate-selected,0) AS fn_mod,
        t.hard,0 AS hard,
        t.hard*selected AS tp_hard,
        GREATEST(t.hard-selected,0) AS fn_hard

        FROM true_alerts t LEFT OUTER JOIN dispnnn v
        ON (t.alert_id = v.alert_no)

        ORDER BY alert_id;"
```

The calculations used to create the scores of TP, FP, TN, and FN were designed to get the correct answer to each score based upon the following truth table (Table A-1). The calculations are complex because values of 0 and 1 were needed in order to obtain their sum; values of True and False would have made it harder to get the sums.

Table A-1 Truth table - scores

True_Alert	Selected	TP	FP	TN	FN
0	**0**	0	0	1	0
0	**1**	0	1	0	0
1	**0**	0	0	0	1
1	**1**	1	0	0	0

After creating the temporary view *dispnnnv*, which has the scores for every alert ID for 1 trial, the summations of the scores are inserted into the table *results_summary* by the following SQL insertion (also invoked in the Python program *create_summary_main.py*).

```
"INSERT INTO results_summary
        (org , subject , display , time_order ,
        true_alerts , selected ,
        tp , fp , tn , fn ,
        f1_score ,
        easy , tp_easy , fn_easy ,
        mod , tp_mod , fn_mod ,
        hard , tp_hard , fn_hard)

        VALUES (
        org, subject, display, time_order,
        (SELECT SUM(true_alert) from dispnnnv),
        (SELECT  SUM(selected) from dispnnnv),
        (SELECT SUM(tp) from dispnnnv),
        (SELECT  SUM(fp) from dispnnnv),
        (SELECT  SUM(tn) from dispnnnv),
        (SELECT  SUM(fn) from dispnnnv),
        (SELECT
2*(SUM(tp))/(2*(SUM(tp))+SUM(fp)+SUM(fn)) from
dispnnnv),
        (SELECT SUM(easy) from dispnnnv),
        (SELECT  SUM(tp_easy) from dispnnnv),
        (SELECT  SUM(fn_easy) from dispnnnv),
        (SELECT SUM(mod) from dispnnnv),
        (SELECT SUM(tp_mod) from dispnnnv),
        (SELECT SUM(fn_mod) from dispnnnv),
        (SELECT SUM(hard) from dispnnnv),
        (SELECT SUM(tp_hard) from dispnnnv),
        (SELECT SUM(fn_hard) from dispnnnv)
        );
```

After this process has been done for all trials, the intermediate table *results_summary* is complete.

A-2.4 Creation of Table *results_summary_plus*

The table *results_summary_plus* contains all of the columns in *results_summary* plus these additional columns:

- `recall       = TP / (TP + FN)  =  Percent of true alerts found`
- `Precision    = TP / (TP + FP)  =  False alarm rate`
- `recall_easy`
- `recall_mod`
- `recall_hard`

This table is built by the Python program *create_summary_plus.py*. Then 1 more field, *elapsed_time*, is added by another Python program, *add_comp_time_main.py*. The first program, *create_summary_plus.py*, reads each line of *results_summary*, calculates the additional fields from existing fields, and adds each new line to the new table, *results_summary_plus*. The second program, *add_comp_time_main.py*, traverses the directory structure and finds all of the original text input files, reading the start, stop, and pause times from them. The time fields are parsed, and the total elapsed time is calculated and inserted into *results_summary_plus*. The Python programs are listed in Appendix B; they include the SQL statements that are built from the SQL query information and the text parser.

After all construction, table *results_summary_plus* has 145 rows, 1 for each subject/display trial. Its columns are as follows:

- *org* — *Organization of the subject – ARL or MSU*
- *subject* — *Subject ID*
- *display* — *Display type (PC, Nodelink, or Table)*
- *time_order* — *For this subject, order of this display type – first, second, or third*
- *completion_time* — *Elapsed time for this trial, not including pause time*
- *tp* — No. of TP selections
- *fp* — No. of FP selections
- *tn* — No. of TN selections
- *fn* — No. of FN selections

- *Recall* *TP / (TP + FN)*
- *Precision* *TP / (TP + FP)*
- *f1_score* 2*TP / (2*TP + FP + FN)
- *tp_easy* No. of easy TP selections
- *fn_easy* No. of easy FN selections
- *recall_easy* *TP_easy / (TP_easy + FN_easy)*
- *tp_mod* No. of moderate TP selections
- *fn_ mod* No. of moderate FN selections
- *recall_ mod* *TP_ mod / (TP_ mod + FN_ mod)*
- *tp_hard* No. of hard TP selections
- *fn_ hard* No. of hard FN selections
- *recall_ hard* *TP_ hard / (TP_ hard + FN_ hard)*

A SQL View was also created into *results_summary_plus*, called *seven_scores*. This view shows summaries useful for analysis. It reduces the display to the following columns: *org, subject, display, completion_time, tp, fp, tn, fn, recall, precision, f1_score*

These are the working copies of the result scores for all trials. They will be used extensively in analysis.

A-3 Correlations and Plots

The *t* tests and *F* tests have been researched and programmed for use in comparing the means of different groups of result data. A plot capability has also been installed and tested. A few examples have been run for demonstration purposes; they are presented here.

A-3.1 Compare Two Means Using Student's *t* Test

The function *compare_means_t*, developed for this analysis, returns the *t* value from the *t* distribution as a function of the 2 groups' standard deviations and degrees of freedom. It also returns the crossover value of *t* at which one can conclude that the two means are different, within a particular confidence level.[2] The confidence

[2]Scipy.org. Statistics package for Python [accessed 2014 Dec 15]. http://docs.scipy.org/doc/scipy-0.14.0/reference/tutorial/stats.html.

level desired is an input to the function. The function also combines the *t* value and the crossover value to return a Boolean value of whether the 2 means are statistically different or not—DIFFERENT or NOT DIFFERENT. This function is in the Python module *functions.py*, which is listed in Appendix B.

The *t* test function has been written, for demonstration purposes, to write results directly into a SQL table, *means_compare*. Table A-2 is a listing of these results.

Table A-2 SQL Table *means_compare*

SQL table means_compare –

research=> select * from means_compare;

mc_id	field	group1	mean1	plusminus1	sigma1	n1	group2	mean2	plusminus2	sigma2	n2	p_value	t_value	t_threshold	different	confidence
7	tp	org='MSU' and display='table'	23.5	3.09	7.62	26	org='ARL' and display='table'	24.78	3.57	8.25	23	41.8	0.661	2.01	NOT DFRNT	0.95
8	tp	org='MSU' and display='pc'	23.35	4.73	11.72	26	org='ARL' and display='pc'	20.25	4.87	10.7	21	63.4	0.9188	2.01	NOT DFRNT	0.95
9	tp	org='MSU' and display='nodelink'	23.08	3.93	9.72	26	org='ARL' and display='nodelink'	25.67	4.99	11.55	23	62.2	0.89032	2.01	NOT DFRNT	0.95
10	fp	org='MSU' and display='table'	29.51	5.25	20.41	26	org='ARL' and display='table'	17.09	7.22	16.7	23	97.7	2.34786	2.01	DIFFERENT	0.95
11	fp	org='MSU' and display='pc'	34.23	10.16	25.15	26	org='ARL' and display='pc'	30.43	12.11	26.61	21	37.2	0.48794	2.01	NOT DFRNT	0.95
12	fp	org='MSU' and display='nodelink'	29.19	7.55	19.68	26	org='ARL' and display='nodelink'	15.74	7.08	16.37	23	98.9	2.63122	2.01	DIFFERENT	0.95
13	tn	org='MSU' and display='table'	68.19	5.25	20.41	26	org='ARL' and display='table'	80.91	7.22	16.7	23	97.7	2.34786	2.01	DIFFERENT	0.95
14	tn	org='MSU' and display='pc'	63.77	10.18	25.15	26	org='ARL' and display='pc'	67.57	12.11	26.61	21	37.2	0.48794	2.01	NOT DFRNT	0.95
15	tn	org='MSU' and display='nodelink'	68.81	7.55	19.68	26	org='ARL' and display='nodelink'	82.26	7.08	16.37	23	98.9	2.63122	2.01	DIFFERENT	0.95
16	fn	org='MSU' and display='table'	18.5	3.08	7.62	26	org='ARL' and display='table'	17.22	3.57	8.28	23	41.6	0.531	2.01	NOT DFRNT	0.95
17	fn	org='MSU' and display='pc'	18.48	4.73	11.72	26	org='ARL' and display='pc'	21.71	4.87	10.7	21	63.4	0.9198	2.01	NOT DFRNT	0.95
18	fn	org='MSU' and display='nodelink'	18.92	3.95	9.72	26	org='ARL' and display='nodelink'	16.13	4.99	11.58	23	62.2	0.89032	2.01	NOT DFRNT	0.95
19	recall	org='MSU' and display='table'	0.38	0.07	0.18	26	org='ARL' and display='table'	0.59	0.09	0.2	23	40.3	0.53252	2.01	NOT DFRNT	0.95
20	recall	org='MSU' and display='pc'	0.56	0.11	0.28	26	org='ARL' and display='pc'	0.48	0.12	0.28	21	63.7	0.91906	2.01	NOT DFRNT	0.95
21	recall	org='MSU' and display='nodelink'	0.55	0.09	0.23	26	org='ARL' and display='nodelink'	0.42	0.12	0.27	23	63.2	0.90909	2.01	NOT DFRNT	0.95
22	precision	org='MSU' and display='table'	0.5	0.07	0.18	26	org='ARL' and display='table'	0.67	0.1	0.24	23	99.1	2.72591	2.01	DIFFERENT	0.95
23	precision	org='MSU' and display='pc'	0.48	0.1	0.24	26	org='ARL' and display='pc'	0.5	0.15	0.32	21	22	0.2814	2.01	NOT DFRNT	0.95
24	precision	org='MSU' and display='nodelink'	0.5	0.1	0.24	26	org='ARL' and display='nodelink'	0.7	0.1	0.23	23	99.6	3.05415	2.01	DIFFERENT	0.95
25	f1_score	org='MSU' and display='table'	0.5	0.06	0.14	26	org='ARL' and display='table'	0.6	0.08	0.18	23	96.3	2.1601	2.01	DIFFERENT	0.95
26	f1_score	org='MSU' and display='pc'	0.46	0.08	0.2	26	org='ARL' and display='pc'	0.45	0.11	0.24	21	5.9	0.07422	2.01	NOT DFRNT	0.95
27	f1_score	org='MSU' and display='nodelink'	0.48	0.08	0.19	26	org='ARL' and display='nodelink'	0.6	0.1	0.25	23	93	1.85772	2.01	NOT DFRNT	0.95
28	recall_easy	org='MSU' and display='table'	0.68	0.18	0.44	26	org='ARL' and display='table'	0.81	0.15	0.34	23	71.8	1.08086	2.01	NOT DFRNT	0.95
29	recall_easy	org='MSU' and display='pc'	0.57	0.18	0.45	26	org='ARL' and display='pc'	0.66	0.22	0.48	21	57.1	0.79796	2.01	NOT DFRNT	0.95
30	recall_easy	org='MSU' and display='nodelink'	0.69	0.18	0.46	26	org='ARL' and display='nodelink'	0.83	0.16	0.38	23	71.9	1.09046	2.01	NOT DFRNT	0.95
31	recall_mod	org='MSU' and display='table'	0.45	0.16	0.4	26	org='ARL' and display='table'	0.62	0.14	0.32	23	89.9	1.65809	2.01	DIFFERENT	0.95
32	recall_mod	org='MSU' and display='pc'	0.43	0.16	0.41	26	org='ARL' and display='pc'	0.44	0.19	0.42	21	4.7	0.05887	2.01	NOT DFRNT	0.95
33	recall_mod	org='MSU' and display='nodelink'	0.73	0.18	0.44	26	org='ARL' and display='nodelink'	0.83	0.16	0.38	23	56.9	0.79436	2.01	NOT DFRNT	0.95
34	recall_hard	org='MSU' and display='table'	0.36	0.08	0.19	26	org='ARL' and display='table'	0.52	0.11	0.26	23	40.1	0.58061	2.01	NOT DFRNT	0.95
35	recall_hard	org='MSU' and display='pc'	0.37	0.12	0.31	26	org='ARL' and display='pc'	0.5	0.12	0.27	21	62.9	0.93589	2.01	NOT DFRNT	0.95
36	recall_hard	org='MSU' and display='nodelink'	0.5	0.09	0.22	26	org='ARL' and display='nodelink'	0.55	0.13	0.3	23	48.9	0.66574	2.01	NOT DFRNT	0.95

(30 rows)

The results for the F1 score are highlighted, because F1 is a balance between Recall and Precision. The ARL analysts were better at 95% confidence using the table display, which is expected because that is similar to the display they use every day. However, using the other 2 displays, they are not better to a 95% level. In addition, using the node-link display, ARL analysts are better to approximately 90% confidence—the *t* value is 1.86 and the threshold for 95% is 2.01.

A-3.2 Compare Multiple Means using *F* Test

The Python function *compare_means_f* accepts all of the distributions being compared, and it outputs the *F* statistic, the *P* value on the *F* distribution, and the *F* distribution value threshold for meeting the desired confidence level.[2] This function is in the Python module *functions.py*, listed in Appendix B.

An example from the trial data has been run, comparing the F1 scores for each subject's first, second, and third trial. Following is the output log from the Python run:

```
ARL - In compare_means_f - there are 3 datasets, with 67 total samples.
means = [ 0.55, 0.54, 0.57 ] - They are NOT different with 95%
confidence.
F = 0.063, F must be > crossover value of 3.14

MSU - In compare_means_f - there are 3 datasets, with 78 total samples.
```

```
means = [ 0.46, 0.52, 0.46 ] - They are NOT different with 95%
confidence.
F = 1.156, F must be > crossover value of 3.119
```

One type of plot has been done (shown in Figs. 4 and 5 in the main report) as an example of the plotting capability. There are many more plot types that can be used in future analyses. The plot prepared is a scatter plot of F1 score, separated by the trial order for each subject—first, second, or third. The means, the 95% confidence interval for the means, and a trend line are superimposed on the scatter plot. These plot were prepared using the Python main program, *order_plots.py*, which is listed in Appendix B.

INTENTIONALLY LEFT BLANK.

Appendix B. Python Programs

The Python programs are listed here, in the order in which they were originally used, which corresponds to the order in which they were described in the body of this report and in Appendix A.

B-1 process_main.py

The Python program *process_main.py* parses the text files containing the subject's selections, picking out all alerts and associating each alert with its group and comment. The program also reads the start time and calculates the elapsed time from the start, stop, and pause times. The program inserts each selection into the SQL table *subject_choices*.

```
'''
Created on Oct 23, 2014

@author: rastrom
'''
import sys
import os
import psycopg2
import sql_connect
import process

if __name__ == '__main__':
    print "Enter process_main - Read survey results into TABLE subject_choices"
    # Set up SQL connection
    conn = sql_connect.connect()
    if conn == -1:
        print "main: connection call failed. Exiting."
        sys.exit(-1)
    cur = conn.cursor()
    total = 0
    # For each results file, get ORG, SUBJECT, and DISPLAY from Dir path and file
name
    #     Then read and parse each file (parse_results)
    #    Then insert data into SQL (insert_results)
    filebase = "/home/rastrom/w00_cognition/FY15/Data"
    inputdata = os.listdir(filebase)
    for org in ['ARL','MSU']: # ARL analysts or Morgan State students
        dirs = os.listdir(filebase + '/' + org)
        for direct in dirs:  # '101' , '112' etc. - represents 'subject' number
            if direct.find('.') == -1:
                subject = int(direct)
                path = filebase + '/' + org + '/' + direct + '/'
                files = os.listdir(path)
                for infile in files:
                    count = 0
                    if (infile.find('table') >= 0 and infile.find('~') == -1): #
Avoid the editing residue of form name.txt~
                        display = 'table'
                        struct = process.parse_results(path + infile)    # See
process.py comments for dictionary 'struct' layout.
                        count = process.insert_results(cur, org, subject, display,
struct)
                        if count > 0:
                            print 'Inserted ' + str(count) + ' lines from ' +
infile
                    elif (infile.find('nodelink') >= 0 and infile.find('~') == -
1):
                        display = 'nodelink'
                        struct = process.parse_results(path + infile)    # See
process.py comments for dictionary 'struct' layout.
```

```
                        count = process.insert_results(cur, org, subject, display,
struct)
                        if count > 0:
                            print 'Inserted ' + str(count) + ' lines from ' +
infile
                    elif (infile.find('pc') >= 0 and infile.find('~') == -1):
                        display = 'pc'
                        struct = process.parse_results(path + infile)    # See
process.py comments for dictionary 'struct' layout.
                        count = process.insert_results(cur, org, subject, display,
struct)
                        if count > 0:
                            print 'Inserted ' + str(count) + ' lines from ' +
infile
                    # End if-elif-elif
                    total = total + count
                # End for infile
            # End if direct.find
        # End for direct
    # End for org
    conn.close()
    print "Exit read_results_main - total insertions = " + str(total)
# end main
```

End process_main.py

B-2 process.py

process.py contains parsing routines called by *process_main.py*.

```
'''
Created on Oct 23, 2014

@author: rastrom

intake - data intake, from results file in .../FY15/Data/ORG/subjectNNN/idNNN-
{nodelink/pc/table}.txt
                      -----------------

        where ORG is one of the two organizations of subjects - ARL or MSU
                SUBJECT is the subject ID number
                and {nodelink/pc/table} are the three different display types.

                e.g. ... /FY15/Data/ARL/101/id101-table.txt  for ARL subject 101
with the 'table' format type.

File format:
------------
{ alerts chosen } \n Time: start: T end T \n [ Paused: T - T ] EOF   ('Paused'
line is optional)

Shortened example:
------------------------------------------------------------
{"1":["6","vulnerability"],"2":["10","EXTREMELY
OLD"],"3":["13","14","16","19","34","38","43","49","FTP scanning"]}
Time: start: 2:44:58 PM, end: 3:00:08 PM
Paused: 2:45:17 PM - 2:46:10 PM,2:54:35 PM - 2:54:46 PM
-------------------------------------------------------------

OUTPUT - dictionary struct = { "start_time":string, "elapsed_time":minutes,
"alerts":[list of alert groups] }
            where [list] = [ {"name":N, "comment":C, "alerts":[N,M,...] } , {...}
, ...]
                    with each group of alerts having its own dict with name,
comment, and alerts list.

'''
```

```
from datetime import datetime, timedelta  # Not necessary


def parse_results(infile):
    # Declare return structure with null entries
    struct = {"alerts":[], "start_time":'01:01:01 AM', "elapsed":0}
    timedata = '01:01:01 AM'
    if infile.find('~') >= 0:
        return struct;
    inputdata = open(infile)
    contents = ""
    for line in inputdata:
        contents = contents + line

    lines = contents.split("\r\n") # 0 = alerts, 1 = Time start / stop, 2 = Time
Paused (if present).
    #                              # There is occasionally a line [3] -
"Submitted: {list of times}". Ignored.
    alerts = lines[0]
    timedata = lines[1].replace("\r","").replace("\n","") # 175-table, 678-table
each have extra "\n" at end.
    pausedata = ""
    if len(lines) > 2:
        pausedata = lines[2]
    time_values = parse_times(timedata, pausedata) # Returns start_time
(datetime.time), elapsed (float - minutes)

    # Initialize result return dictionary
    struct = {"alerts":[], "start_time":time_values[0], "elapsed": time_values[1]}

    # Parse alerts into structure
    # Remove end point brackets
    alert1 = alerts.replace("{","")
    alert2 = alert1.replace("}","")
    # There might be no alerts chosen - alert2 would now be null string
    if alert2 == "":
        print "No alerts in " + infile
        return struct
    # Separate alerts into Groups using " [ ] "
    groups = alert2.split("],")
    #print "groups = "
    #print groups
    for g in groups:
        components = g.split(":[")
        #print "components ="
        #print components
        gname = components[0]             # group name chosen by subject
        groupname = gname.replace('"','') # groupname is text string, typically
'1', '2', etc.
        galerts = components[1].replace("]","")
        #print "group "+groupname+" = " + galerts
        # first split off comment at end
        quotesplit = galerts.split('"') # will make a mess of alert numbers if
they are of form "N","N",...
        n = len(quotesplit)
        #print "len = " + str(n) + " and quotesplit = "
        #print quotesplit
        groupcomment = quotesplit[n-2] # next-to-last one is the comment - the
last one is '' after the last quote.
        #print "comment = '" + groupcomment + "'"
        #print "galerts = '" + galerts + "'"
        replacement = ',"'+groupcomment+'"'
        #index =
        alertsonly = galerts.replace(replacement,'') # Strip off comment at end
        #print "alsertsonly = " + alertsonly

        # Re-split - new alertsplit has only alert numbers (with or without
surrounding quotes)
        alertsplit = alertsonly.split(',')
        #print "alertsplit = "
        #print alertsplit
```

```
        groupalerts = [] # initialize list of alerts chosen by this subject in
this group
        for alertno in alertsplit:
            try:
                alert = int(alertno.replace('"','')) # Strip "" and cast as
integer (some are "N" and others are just N )
                groupalerts.append(alert)
            except:
                print "### Data casting error. Failed to cast '"\
                + alertno.replace('"','') +"' -  Infile " + infile + " -  Group
text = " + galerts
                continue

        # end for alertno
        #print groupalerts
        # Build small dict with group name, comment, list
        groupstruct = {"name":groupname, "comment":groupcomment,
"alerts":groupalerts} # One per subject group - name, comment, alert list
        #print groupstruct
        struct["alerts"].append(groupstruct)
    # End for g
    inputdata.close()

    return struct
# end parse_results

'''
Method parse_times
Input: Takes the last lines of the input file (after the alerts).
Line[1] is start - end times;
Line[2] (Optional) is Pause times.

Output: returns start time (datetime.time) and total elapsed time (integer -
minutes)
'''
def parse_times(timedata, pausedata):

    # Start & end first - timedata line
    index = timedata.find("Time: start: ")
    partial = timedata[index+13:]
    times = partial.split(",")
    start = times[0]          # Format = 'hh:mm:ss PM' - this is accepted into a
SQL Time field.
    ampm = True # Flag: Time value has hh:mm:ss PM (or AM) in it. some do, some
don't.
    amind = start.find("AM")
    pmind = start.find("PM")
    if amind == -1 and pmind == -1:
        ampm = False
    start_dt = datetime.now() # scope issue - define here
    end_dt = datetime.now()   # scope issue - define here
    try:
        if ampm:
            start_dt = datetime.strptime("01/01/2001 "+ str(start),"%m/%d/%Y
%I:%M:%S %p")
        else:
            start_dt = datetime.strptime("01/01/2001 "+ str(start),"%m/%d/%Y
%H:%M:%S")
        end_dt = start_dt
    except:
        print "strptime failure - time data = " + timedata
        return start, 0
    endtext = times[1]
    index = endtext.find("end: ")
    endtime = endtext[index+5:]
    try:
        if ampm:
            end_dt = datetime.strptime("01/01/2001 "+ str(endtime),"%m/%d/%Y
%I:%M:%S %p")
        else:
```

```
            end_dt = datetime.strptime("01/01/2001 "+ str(endtime),"%m/%d/%Y
%H:%M:%S")
    except:
        print "strptime failure - time data = " + timedata
        return start, 0
    if end_dt < start_dt:
        print "Elapsed time < 0 - start time = " + start
        return start, 0
    elapsedtemp = end_dt - start_dt
    #print elapsedtemp
    elapsedsec = elapsedtemp.seconds
    #print start,endtime
    # Now subtract pauses from elapsed time - done in seconds
    if pausedata != "":
        ind = pausedata.find("Paused: ")
        if ind != -1:  # There are pauses
            pausetimes = pausedata[ind+8:]
            pauses = pausetimes.split(",")
            for pause in pauses:
                #print pause
                times = pause.split("-")
                start = times[0].strip()          # Format = 'hh:mm:ss PM' - this
is accepted into a SQL Time field.
                endtime = times[1].strip()
                try:
                    if ampm:
                        start_dt = datetime.strptime("01/01/2001 "+
str(start),"%m/%d/%Y %I:%M:%S %p")
                        end_dt = datetime.strptime("01/01/2001 "+
str(endtime),"%m/%d/%Y %I:%M:%S %p")
                    else:
                        start_dt = datetime.strptime("01/01/2001 "+
str(start),"%m/%d/%Y %H:%M:%S")
                        end_dt = datetime.strptime("01/01/2001 "+
str(endtime),"%m/%d/%Y %H:%M:%S")
                    # End if-else
                except:
                    print "strptime failure - pause data = " + pausetimes
                    return start,0
                # End try-except
                if end_dt < start_dt:
                    print "Pause time < 0 - pause data = " + pausetimes
                    return start, 0
                elapsedtemp = end_dt - start_dt
                pausesec = elapsedtemp.seconds
                #print start,endtime,pausesec
                #print "Pause of " + str(pausesec) + " seconds subtracted."
                elapsedsec = elapsedsec - pausesec
            # End for pause
        # End if ind
    # End if pausedata
    elapsed = round(float(elapsedsec)/60.0,2)
    return start, elapsed
# End parse_times

'''
Method insert_results
Takes one data file, already parsed into 'struct', and inserts that data into
table 'subject_choices'

struct - layout described above in the header to 'parse_results'

The subject group (ARL or MSU), the subject ID, and the display type (nodelist,
pc, table) are part of the file name or
        directory path, not in the file text that is parsed by parse_results.
        They have to be input to 'insert_results' as calling parameters.
'''

def insert_results(cur,subject_org,subject,display,struct):
    start_time = struct["start_time"]
    '''
```

```
    # TEST ONLY
    alert_no = 6
    a_group = "1"
    group_comment = "Common ..."
    # END TEST ONLY
    '''

    # Loop through each group of selected alerts, and insert each alert into table
subject_choices.
    count = 0
    dupcount = 0
    errcount = 0
    groups = struct["alerts"]
    for group in groups:
        a_group = group["name"]
        group_comment = group["comment"]
        for alert_no in group["alerts"]:
            # Check if already in
            dupincr = 0
            slct = "select count(*) from subject_choices where (subject = '" +
str(subject) + "' and display = '" + display + "' and alert_no = " + str(alert_no)
+ ");"
            #print slct
            cur.execute(slct)
            rows = cur.fetchall()
            for row in rows:
                if row[0] > 0:
                    #print "######################## Duplicate insertion - " +
slct
                    dupincr = 1 # Duplication flag

            # End for - done checking for duplication
            if dupincr == 1:
                dupcount = dupcount + 1
                continue # End for this alert no
            group_comment_a = group_comment.replace("'","''") # Escape any ' chars
            ins = "INSERT INTO subject_choices
(subject_org,subject,display,start_time,alert_no,a_group, group_comment) VALUES
('"\
            + subject_org + "','" + str(subject) + "','" + display + "','"\
            + start_time + "','" + str(alert_no) + "'," + a_group + ",'" +
group_comment_a + "');"
            try:
                cur.execute(ins)
                count = count + 1
            except:
                print "################### Error in INSERT - " + str(alert_no) + "
- " + ins
                cur.execute("COMMIT;")
            # End try-except
        # End for alert_no
    # End for group - End of loop through alert groups
    if count > 0:
        cur.execute("COMMIT;")
    if dupcount > 0:
        print "################ " + str(dupcount) + " Duplicate entries attempted
into subject " + str(subject) + ", display type " + display
    return count
# end insert_results
```

End process.py

B-3 set_order_main.py

set_order_main.py performs a SQL query to pick out the 3 start times for each subject, sorts them in order, and then inserts “1”, “2’, or “3” into the field *time_order* in each row of the table *subject_choices*.

```
'''
Created on Nov 3, 2014

@author: rastrom
'''

import sys
import os
import psycopg2
import sql_connect
import process

if __name__ == '__main__':
    print "Enter set_order_main - For each distinct subject-display-time, set
time_order to 1, 2, or 3."
    # Set up SQL connection
    conn = sql_connect.connect()
    if conn == -1:
        print "main: connection call failed. Exiting."
        sys.exit(-1)
    cur = conn.cursor()
    cur.execute("SELECT DISTINCT subject FROM subject_choices;")
    count = 0
    subjects = cur.fetchall()
    for subject in subjects:
        count = count + 1
        print "Begin subject " + subject[0]
        temptable = "times" + subject[0]    # Create temporary table timesNNN
        cur.execute("create temporary table " + temptable + " (start_time
time,display text,time_order SERIAL PRIMARY KEY);")
        cur.execute("COMMIT;")
        cur.execute("insert into " + temptable\
                    + " (start_time,display) (select distinct on (start_time,
display) start_time,display from subject_choices where subject = '"\
                    + subject[0] +"' order by start_time);") # The serial primary
key, auto-filled-in, provides the ordering 1,2,3.
        cur.execute("COMMIT;")
        cur.execute("select * from " + temptable + ";")
        lines = cur.fetchall()
        for line in lines:
            print line
        cur.execute("update subject_choices c set time_order = (select time_order
from "\
                    + temptable +" t where t.display = c.display) where subject =
'" + subject[0] + "';")
        cur.execute("DROP TABLE " + temptable + ";")
        cur.execute("COMMIT;")
    print "End of subjects. Count = " + str(count)
    conn.close()
# end main
```

End set_order_main.py

B-4 create_summary_main.py

This Python program creates a new table row in the table *results_summary* for each trial (a trial is 1 subject using 1 display type).

```
'''
Created on Nov 6, 2014

@author: rastrom

Creates the overall summary of the alert sessions with the subjects.

Inputs are:
            1. The table true_alerts,
               which has a row for each of the 140 alerts shown.
               For each alert, it shows true_alert = 1 or 0 (yes or no).
               For each true alert, the table also shows whether the alert was
easy, moderate, or hard to identify.
            2. The table subject_choices,
               which has a row for each alert (140 of them) for each subject, for
each display type (7,227 rows).
               For each subject/display/alert, it shows their answer.

Intermediate temporary views are:

            1. table101 (e.g.) - view of all the rows from subject_choices for one
analyst and one display type.
            2. table101v (e.g.) - view created by JOIN of table101 and
true_alerts, which has one row for each alert (140)
               For each subject/display/alert, it shows their answer,
               and whether that answer is a true positive (TP), true negative
(TN), false positive (FP), or false negative (FN)

Output is:  table results_summary, which has one row for each subject/display (145
rows).
            For each one, it shows the count of alerts selected, along with TP,
TN, FP, and FN counts; the F1 score;
            and counts of TP and FN for the easy, moderate, and hard true alerts.
'''

import sys
import os
import psycopg2
import sql_connect
import process

def get_numeric(item): # Used in sorting rows
    return int(item[0])

if __name__ == '__main__':
    print "Enter create_summary_main - For each distinct subject-display, add one
row with results."
    # Set up SQL connection
    conn = sql_connect.connect()
    if conn == -1:
        print "main: connection call failed. Exiting."
        sys.exit(-1)
    cur = conn.cursor()

    # First set up loop parameters - display and subject
    cur.execute("SELECT DISTINCT ON (subject,display) subject, display,
subject_org, time_order from subject_choices ORDER BY subject")
    rowsin = cur.fetchall()
    rows = sorted(rowsin, key=get_numeric) # Do the sort on the integer value of
"subject"
    #print rows
    #print "subj,disp count = " + str(len(rows))
    for subject_display in rows:
        subject = subject_display[0]
        display = subject_display[1]
        org = subject_display[2]
        time_order = subject_display[3]
        print "s-d-o-t = " + subject + ", " + display + ", " + org  + ", " +
str(time_order)

        # Process this subject-display pair
```

```
        #
        # First create temp view dispnnn (e.g. table101)
        c_dispnnn = "CREATE OR REPLACE TEMPORARY VIEW dispnnn AS (SELECT
subject_org,subject,time_order,alert_no,a_group,group_comment FROM subject_choices
" + \
        "WHERE subject = '" + subject + "' AND display = '" + display + "');"
        #print c_dispnnn
        cur.execute(c_dispnnn)
        cur.execute("COMMIT;")

        # Cteate dispnnnv (e.g. table101v)
        c_dispnnnv = "CREATE OR REPLACE TEMPORARY VIEW dispnnnv AS SELECT
t.alert_id, v.subject_org AS org, v.subject, v.time_order, "+\
        " COALESCE(t.true_alert,0) AS true_alert,"+\
        " COALESCE((v.alert_no/v.alert_no),0) AS selected, "+\
        " COALESCE((t.true_alert*(v.alert_no/v.alert_no)),0) AS tp, "+\
        " COALESCE(GREATEST(COALESCE((v.alert_no/v.alert_no))-
COALESCE(t.true_alert,0)),0) AS fp, "+\
        " -GREATEST(t.true_alert,COALESCE((v.alert_no/v.alert_no),0))+1 AS tn, "+\
        " COALESCE(GREATEST(COALESCE(t.true_alert,0)-
COALESCE((v.alert_no/v.alert_no),0),0)) AS fn, "+\
        " COALESCE(t.easy,0) AS easy, COALESCE((t.easy*(v.alert_no/v.alert_no)),0)
AS tp_easy, "+\
        " COALESCE(GREATEST(COALESCE(t.easy,0)-
COALESCE((v.alert_no/v.alert_no),0),0)) AS fn_easy, "+\
        " COALESCE(t.moderate,0) AS mod, "+\
        " COALESCE((t.moderate*(v.alert_no/v.alert_no)),0) AS tp_mod, "+\
        " COALESCE(GREATEST(COALESCE(t.moderate,0)-
COALESCE((v.alert_no/v.alert_no),0),0)) AS fn_mod, "+\
        " COALESCE(t.hard,0) AS hard, "+\
        " COALESCE((t.hard*(v.alert_no/v.alert_no)),0) AS tp_hard, "+\
        " COALESCE(GREATEST(COALESCE(t.hard,0)-
COALESCE((v.alert_no/v.alert_no),0),0)) AS fn_hard "+\
        " FROM true_alerts t LEFT OUTER JOIN dispnnn v ON (t.alert_id =
v.alert_no) "
        " ORDER BY alert_id;"
        #print c_dispnnnv
        cur.execute(c_dispnnnv)
        cur.execute("COMMIT;")

        c_insert = "INSERT INTO results_summary " + \
        " (org , subject , display , time_order ," + \
        " true_alerts , selected ," + \
        " tp , fp , tn , fn ," + \
        " f1_score ," + \
        " easy , tp_easy , fn_easy ," + \
        " mod , tp_mod , fn_mod ," + \
        " hard , tp_hard , fn_hard)" + \
        " VALUES ( '" + org +"', " + str(subject) +", '" + display +"'," +
str(time_order) + ", " + \
        " (SELECT SUM(true_alert) from dispnnnv), (SELECT  SUM(selected) from
dispnnnv)," + \
        " (SELECT SUM(tp) from dispnnnv), (SELECT  SUM(fp) from dispnnnv)," + \
        " (SELECT  SUM(tn) from dispnnnv), (SELECT  SUM(fn) from dispnnnv)," + \
        " (SELECT ROUND(
2*(SUM(tp))::NUMERIC/(2*(SUM(tp))+SUM(fp)+SUM(fn))::NUMERIC,2 ) from dispnnnv)," +
\
        " (SELECT SUM(easy) from dispnnnv), (SELECT  SUM(tp_easy) from dispnnnv),
(SELECT  SUM(fn_easy) from dispnnnv)," + \
        " (SELECT SUM(mod) from dispnnnv), (SELECT SUM(tp_mod) from dispnnnv),
(SELECT SUM(fn_mod) from dispnnnv)," + \
        " (SELECT SUM(hard) from dispnnnv), (SELECT SUM(tp_hard) from dispnnnv),
(SELECT SUM(fn_hard) from dispnnnv) );"
        #print c_insert

        try:
            cur.execute(c_insert)
        except:
            print "Insertion failed (probably duplication) - " + str(subject) + ",
" + display
```

```
        cur.execute("DROP VIEW dispnnnv;")
        cur.execute("DROP VIEW dispnnn;")
    # End for subject-display - All results have been inserted.
    print "End of create_summary_main"
    conn.close()
# End main
```

End create_summary_main.py

B-5 create_summary_plus_main.py

This program adds the statistics Recall and Precision to the results table, plus Recall for easy, moderate, and hard subcategories of true alerts. It creates a new table, *results_summary_plus*.

```
'''
Created on Nov 12, 2014

@author: rastrom

create_summary_plus_main

Inserts all 145 data rows into an upgrade over the table results_summary.

Input: table results_summary

Output: similar table results_summary_plus, which has all columns that
results_summary has, plus the folowing:

    recall      = TP / (TP + FN)
    Precision   = TP / (TP + FP)
    recall_easy
    recall_mod
    recall_hard

'''

import sys
import os
import psycopg2
import sql_connect
import process

if __name__ == '__main__':
    print "Enter create_summary_plus_main - For each distinct subject-display, add
one row with results PLUS."
    # Set up SQL connection
    conn = sql_connect.connect()
    if conn == -1:
        print "main: connection call failed. Exiting."
        sys.exit(-1)
    cur = conn.cursor()

    # First copy all of table results_summary
    cur.execute("SELECT * from results_summary;")
    rows = cur.fetchall()
    for row in rows:
        rsindex     = row[0]
        org         = row[1]
        subject     = row[2]
        display     = row[3]
        time_order  = row[4]
        true_alerts = row[5]
        selected    = row[6]
        tp          = row[7]
        fp          = row[8]
        tn          = row[9]
```

```
        fn          = row[10]
        f1_score    = row[11]
        easy        = row[12]
        tp_easy     = row[13]
        fn_easy     = row[14]
        mod         = row[15]
        tp_mod      = row[16]
        fn_mod      = row[17]
        hard        = row[18]
        tp_hard     = row[19]
        fn_hard     = row[20]
        #print row

        # ADD columns: elapsed time, recall, precision, recall_easy, recall_mod,
recall_hard.

        # First re-parse input files for time information.


        # Prepare INSERT statement for this row
        c_insert = "INSERT INTO results_summary_plus " + \
        " (rsindex, org , subject , display , time_order , " + \
        " true_alerts , selected , tp , fp , tn , fn , " + \
        " recall , precision , f1_score , " + \
        " easy , tp_easy , fn_easy , recall_easy , " + \
        " mod , tp_mod , fn_mod , recall_mod , " + \
        " hard , tp_hard , fn_hard , recall_hard)" + \
        " VALUES ( " + str(rsindex) + ", '" + org +"', " + str(subject) +", '" +
display +"'," + str(time_order) + ", " + \
        str(true_alerts) + ", " + str(selected) + ", " + str(tp) + ", " + str(fp)
+ ", "+ str(tn) + ", " + str(fn) + \
        ", ROUND( (" + str( float(tp) / ( float(tp) + float(fn) ) ) + "),2 ) " +
", ROUND( (" + str( float(tp) / ( float(tp) + float(fp) ) ) + "),2 ), " + \
        str(f1_score) + ", " + str(easy) + ", " + str(tp_easy) + ", " +
str(fn_easy) + \
        ", ROUND( (" + str( float(tp_easy) / ( float(tp_easy) + float(fn_easy) ) )
+ "),2 ), " + \
        str(mod) + ", " + str(tp_mod) + ", " + str(fn_mod) + \
        ", ROUND( (" + str( float(tp_mod) / ( float(tp_mod) + float(fn_mod) ) ) +
"),2 ), " + \
        str(hard) + ", " + str(tp_hard) + ", " + str(fn_hard) + \
        ", ROUND( (" + str( float(tp_hard) / ( float(tp_hard) + float(fn_hard) ) )
+ "),2 ) ) ; "
        #print c_insert
        try:
            cur.execute(c_insert)
        except:
            print "Exception - probable duplication - " + c_insert

    # End for row - done here.
    cur.execute("COMMIT;")
    print "End of create_summary_plus_main"
    conn.close()
# End main
```

End create_summary_plus_main.py

B-6 add_comp_time_main.py

One last column is filled in, for the results table *results_summary_plus*, by this program.

```
'''
Created on Nov 13, 2014

@author: rastrom
'''
```

```
import sys
import os
import psycopg2
import sql_connect
import process

if __name__ == '__main__':
    print "Enter add_comp_time_main - Read & parse survey results, add completion
time to results_summary_plus. "
    # Set up SQL connection
    conn = sql_connect.connect()
    if conn == -1:
        print "main: connection call failed. Exiting."
        sys.exit(-1)
    cur = conn.cursor()
    total = 0
    # For each results file, get ORG, SUBJECT, and DISPLAY from Dir path and file
name
    #     Then read and parse each file (parse_results)
    #    Then insert data into SQL (insert_results)
    filebase = "/home/rastrom/w00_cognition/FY15/Data"
    inputdata = os.listdir(filebase)
    for org in ['ARL','MSU']: # ARL analysts or Morgan State students
        dirs = os.listdir(filebase + '/' + org)
        for direct in dirs:  # '101' , '112' etc. - represents 'subject' number
            if direct.find('.') == -1:
                subject = int(direct)
                path = filebase + '/' + org + '/' + direct + '/'
                files = os.listdir(path)
                for infile in files:
                    count = 0
                    if (infile.find('table') >= 0 and infile.find('~') == -1): #
Avoid the editing residue of form name.txt~
                        display = 'table'
                        struct = process.parse_results(path + infile)    # See
process.py comments for dictionary 'struct' layout.
                        if struct["elapsed"] == 0:
                            print "Elapsed time of 0 - subject,display = " +
str(subject) + ", " + display
                        #print struct
                        else:
                            insert = "UPDATE results_summary_plus SET
completion_time = " + str(struct['elapsed']) + \
                            " WHERE subject = " + str(subject) + " AND display =
'table';"
                            #print insert
                            cur.execute(insert)
                            cur.execute("COMMIT;")
                            count = count + 1
                        # End if-else
                    elif (infile.find('nodelink') >= 0 and infile.find('~') == -
1):
                        display = 'nodelink'
                        struct = process.parse_results(path + infile)    # See
process.py comments for dictionary 'struct' layout.
                        if struct["elapsed"] == 0:
                            print "Elapsed time of 0 - subject,display = " +
str(subject) + ", " + display
                        #print struct
                        else:
                            insert = "UPDATE results_summary_plus SET
completion_time = " + str(struct['elapsed']) + \
                            " WHERE subject = " + str(subject) + " AND display =
'nodelink';"
                            #print insert
                            cur.execute(insert)
                            cur.execute("COMMIT;")
                            count = count + 1
                        # End if-else
                    elif (infile.find('pc') >= 0 and infile.find('~') == -1):
                        display = 'pc'
```

```
                        struct = process.parse_results(path + infile)    # See
process.py comments for dictionary 'struct' layout.
                        if struct["elapsed"] > 0 and struct["alerts"] != []:
                            insert = "UPDATE results_summary_plus SET
completion_time = " + str(struct['elapsed']) + \
                            " WHERE subject = " + str(subject) + " AND display =
'pc';"
                            #print insert
                            cur.execute(insert)
                            cur.execute("COMMIT;")
                            count = count + 1
                        # End if
                    # End if-elif-elif

                    total = total + count
                # End for infile
            # End if direct.find
        # End for direct
    # End for org
    conn.close()
    print "Exit add_comp_time_main - total insertions = " + str(total)
# end main
```

End add_comp_time_main.py

B-7 Statistical Analysis – Means Comparison Routines in functions.py

Functions.py contains 2 mean comparison tests: compare_means_t and compare_means_f.

```
'''
Created on Nov 20, 2014

@author: rastrom
'''
import scipy.stats as stats
from scipy.stats import t
import numpy
'''
# Subroutine to determine the t-statistic of a distribution
# Inputs:
#    n = number of samples
#    std = standard deviation of the samples.
#    confidence = % confidence limit desired, e.g. for 95% or 99% conf level of
interval
#
# Output: half_range = half the length of confidence interval - e.g. if half_range
= 1 then conf.int. = mean +- 1
#
'''
def t_conf_int(n, istd, iconfidence):
    #                         CAUTION: SELECT results from "float" fields are
copied as "Decimal", which
    #                         does not mix with float in arithmetic. Fixed in
functions, not here.
    #                         COUNT ( "n" here) is copied as "long" integer; OK in
arithmetic.
    std = float(istd)
    confidence = float(iconfidence)
    if n < 2:
        print "Bad t_stat call: n,std,conf = " + str(n) + str(std) +
str(confidence)
        return 0.0
    # intv = interval(alpha, df, loc=0, scale=1)     # Endpoints of the range that
contains alpha percent of the distribution
    intv = t.interval(confidence, n-1, loc=0, scale=1)
```

```
    t_val = intv[1]
    #print "t value for " +  str(n) + " samples and confidence interval of " +
str(confidence) + " = " + str(t_val)
    half_range = t_val*( std / numpy.sqrt(n) )
    return half_range
# End t_stat
'''
# compare_means_t -
#
# Subroutine to SELECT 'field' from results_summary_plus for two different groups,
# and calculate & print each group's mean, standard deviation, and 95% plus-minus,
# plus, for the PAIR, DIFFERENT (T or F), t-test t-value, p-value, and t-threshold
for 95%.
#
'''
def compare_means_t(cur, confidence, field, where1, where2):
    #test mean difference - using f1_score, MSU & ARL
    # Read the vectors
    INSERT = True     ##### Parameter, settable here.
    VERBOSE = True    ##### Parameter, settable here.
    sql1 = "SELECT " + field + " FROM results_summary_plus WHERE " + where1 + ";"
    #print sql1 # Deubg only
    cur.execute(sql1)
    rows = cur.fetchall() # len = 78 values of f1 for MSU subjects
    field1 = []
    for row in rows:
        field1.append(float(row[0]))
    field1_avg = numpy.mean(field1)
    field1_stdev = numpy.std(field1)
    field1_len = len(field1)
    sql2 = "SELECT " + field + " FROM results_summary_plus WHERE " + where2 + ";"
    cur.execute(sql2)
    rows = cur.fetchall() # len = 67 values of f1 for ARL subjects
    field2 = []
    for row in rows:
        field2.append(float(row[0]))
    field2_avg = numpy.mean(field2)
    field2_stdev = numpy.std(field2)
    field2_len = len(field2)

    # Calculate 95% confidence interval for the means.
    half_range1 = t_conf_int(field1_len, field1_stdev, confidence)

    half_range2 = t_conf_int(field2_len, field2_stdev, confidence)

    # Call ttest_ind to determine whether the 2 means are statistically different.
    # Result[1] = probability = tail of the t distribution. Lower prob --> more
likely that means are different.
    # Example: if result[1] <= .05 = 5% , then the 2 means are different with 95%
confidence.
    result = stats.ttest_ind(field1, field2, equal_var=False)
    # Calculate confidence interval for hypothesis that the 2 means are different
-
    pct = round(100.0 - result[1]*100.0 , 1)
    # Calculate t stat for len+len
    n = field1_len + field2_len - 1 # Should this be -2 ?
    intv = t.interval(0.95, n-1, loc=0, scale=1)
    tcalc = intv[1]

    # Output format:
    #
    # Field 'field' - 95% confidence interval for means, 2-tailed:
    # "where1" mean = 29.81 +- 8.25 ; sigma = 20.41 ; n = 26 samples
    # "where2" mean = 17.09 +- 7.22 ; sigma = 16.7 ; n = 23 samples
    # P = 97.7%        probabilility of significant difference
    # t value = 2.34;  values over
    # t-dist = 2.01    indicate a significant difference with confidence of 95%

    # Insert into means_compare table
    w1 = where1.replace("'", "''") # escape ' chars in WHERE clause
    w2 = where2.replace("'", "''")
```

```
    if pct < confidence*100.0:
        different = "NOT DFRNT"
    else:
        different = "DIFFERENT"
    means_insert = "INSERT INTO means_compare " + \
    "( field, group1, mean1, plusminus1, sigma1, n1," \
    + " group2, mean2, plusminus2, sigma2, n2," \
    + " p_value, t_value, t_threshold, different, confidence )" \
    + " VALUES ( '" + field +"', '" \
    + w1 +"', " + str( round(field1_avg,2)) + ", " + str(round(half_range1,2)) \
    + ", " + str(round(field1_stdev,2)) + ", " + str(field1_len) + ", '" \
    + w2 + "', " + str( round(field2_avg,2)) + ", " + str(round(half_range2,2)) \
    + ", " + str(round(field2_stdev,2)) + ", " + str(field2_len) + ", " \
    + str(round(100.0-result[1]*100.0,1)) + ", " + str(round(abs(result[0]),5)) +
", " + str(round(tcalc,2)) \
    + ", '" + different + "', " + str(confidence) \
    + " );"
    #print means_insert # Debug only
    if INSERT == True:
        try:
            cur.execute(means_insert)
        except:
            a = "***********"
            print a+"These 2 means in field '" + field + "' are already in the
means_compare table. "+a
        # End try
        cur.execute("COMMIT") # STRANGE lesson learned: "COMMIT" an exception
here, or the next SELECT fails.
    # End INSERT
    # Create a printed report if VERBOSE is True:
    if VERBOSE == True:
        print "Field '" + field + "' - " + str(int(confidence*100)) \
        + "% confidence interval for means, 2-tailed:"
        print "\"" + where1 + "\" - mean = " + str( round(field1_avg,2)) + " +- "
+ str(round(half_range1,2)) \
        + " ; sigma = " + str(round(field1_stdev,2)) + " ; n = " + str(field1_len)
+ " samples"
        print "\"" + where2 + "\" - mean = " + str( round(field2_avg,2)) + " +- "
+ str(round(half_range2,2)) \
        + " ; sigma = " + str(round(field2_stdev,2)) + " ; n = " + str(field2_len)
+ " samples"
        if pct < confidence*100.0:
            print "Means are NOT different with confidence of " +
str(int(confidence*100)) + "%"
        else:
            print "Means ARE different with confidence of " +
str(int(confidence*100)) + "%"
        print "P = " + str(pct) + "%            probability of significant
difference between means\n" \
        + "t value = " + str(round(abs(result[0]),5)) + " ;  values over\n" \
        + "t-dist = "+ str(round(tcalc,2)) + "        indicate a significant
difference with confidence of 95%\n"
    # End VERBOSE
    #md = field2_avg - field1_avg
    #se = numpy.sqrt(field2_stdev**2/(field2_len-1) + field1_stdev**2/(field1_len-
1))
    #tval = md/se
    #print "Directly calculated t stat = " + str(round(abs(tval),5))
    return
# End compare_means_t
'''
# compare_means_F
#
# Method to compare two or more means.
#
# Determines whether ANY mean or means is significantly different from the others.
#        (Passing this F test ["Different"] does NOT tell you which one(s) is/are
different.)
#
# Uses F test. ASSUMPTION : all variances are the same.
#
```

```
# Inputs: confidence, [X1,X2,...] (at least 2 ) , where
#        confidence is the desired confidence limit (e.g. .95 for 95% confidence)
#        [X1,X2,...] (in a list/2d array) are the vectors of values in set 1, 2,
etc.
#
# Outputs: [ F , p-value , f-dist ] , where
#        F is the F statistic for the N sets of data
#        p-value is the p-value on the F distribution CDF
#        f-dist is the crossover value on the CDF for p-value where the means are
DIFFERENT within conf% confidence.
#
# YES / NO - The calling routine must calculate the Yes or No answer to the
question:
#        "Are any of the means different from the others, to a (conf)% confidence
level ??
#
#        DIFFERENT iff p-value > f-dist
'''
def compare_means_f(confidence, inputlist):
    x = inputlist
    print "Enter compare_means_f, conf = " + str(confidence)
    k = 0 # K parameter - number of data sets.
    n = 0 # N parameter - total number of samples
    ni = [] # List of N (number of samples) for each input vector
    for vec in x:
        print len(vec)
        k = k + 1
        n = n + len(vec)
        ni.append(len(vec))
    # End for vec
    print "There are " + str(k) + " datasets, with " + str(n) + " total samples."
    if k < 2:
        print "There are not at least 2 vectors of data! compare_means_f returns
0. ############"
        return [0,0,0]
    if k == 2:
        ft = stats.f_oneway(x[0],x[1])
    elif k == 3:
        ft = stats.f_oneway(x[0],x[1],x[2])
    elif k == 4:
        ft = stats.f_oneway(x[0],x[1],x[2],x[3])
    elif k == 5:
        ft = stats.f_oneway(x[0],x[1],x[2],x[3],x[4])
    elif k == 6:
        ft = stats.f_oneway(x[0],x[1],x[2],x[3],x[4],x[5])
    else:
        print "There are more than 6 vectors of data! Change the code if this is
legitimate. ###############"
        return [0,0,0]
    # End if-elif-else
    F = ft[0]
    pval = ft[1]
    print "f_oneway - F, p :"
    print ft # answer: F stat = 9.27; p-value = .00239.

    fdist = stats.distributions.f.ppf(.95,2,15)
    print "cdf :"
    print fdist

    return [F, pval, fdist]
# End compare_means_f

# End functions
```

End functions.py

B-8 mean_tests.py

Mean_tests.py, as a first exercise of using the t-test, calls the function *compare_means_t* to compare the means of 10 scores from *results_summary_plus*, for 3 display types each. The 30 tests are all comparisons of US Army Research Laboratory (ARL) versus Morgan State University (MSU) test subjects.

```
'''
Created on Dec 3, 2014

@author: rastrom
'''
import sys
import sql_connect
import functions
import numpy
import scipy.stats as s
from scipy.stats import t

# Compares the two means of "field" between MSU & ARL subjects, all display types
def org_display_compare(cur, field, confidence):
    where1 = "org='MSU' and display='table'"
    where2 = "org='ARL' and display='table'"
    functions.compare_means_t(cur, confidence, field, where1, where2)
    where1 = "org='MSU' and display='pc'"
    where2 = "org='ARL' and display='pc'"
    functions.compare_means_t(cur, confidence, field, where1, where2)
    where1 = "org='MSU' and display='nodelink'"
    where2 = "org='ARL' and display='nodelink'"
    functions.compare_means_t(cur, confidence, field, where1, where2)

# End org_display_compare

# Test T statistic
#
# Test PAIRS of mean values, using the t test, and present True or False - are
they different?
#        (within 95% confidence interval) - OR 99%; parameter.
if __name__ == '__main__':
    print "Enter mean_tests\n"
    conn = sql_connect.connect()
    if conn == -1:
        print "main: connection call failed. Exiting."
        sys.exit(-1)
    cur = conn.cursor()
    confidence = .95
    field = "tp"
    org_display_compare(cur, field, confidence)
    field = "fp"
    org_display_compare(cur, field, confidence)
    field = "tn"
    org_display_compare(cur, field, confidence)
    field = "fn"
    org_display_compare(cur, field, confidence)
    field = "recall"
    org_display_compare(cur, field, confidence)
    field = "precision"
    org_display_compare(cur, field, confidence)
    field = "f1_score"
    org_display_compare(cur, field, confidence)
    field = "recall_easy"
    org_display_compare(cur, field, confidence)
    field = "recall_mod"
    org_display_compare(cur, field, confidence)
    field = "recall_hard"
    org_display_compare(cur, field, confidence)
    print "\nEnd mean_tests"
```

```
    conn.close()
# End main
```

B-9 order_plots.py

This program uses SQL queries to get all F1 scores, along with their means and standard deviations. It then calls the *T* test, means_test_t, listed in functions.py, to determine each mean's 95% confidence interval. Then the scatter plot is created, with the means and intervals superimposed.

```
'''
Created on Dec 16, 2014

@author: rastrom
'''

import sys
import sql_connect
import functions
import numpy
import matplotlib.pyplot as plt

if __name__ == '__main__':
    print "Enter order_plots"
    conn = sql_connect.connect()
    if conn == -1:
        print "main: connection call failed. Exiting."
        sys.exit(-1)
    cur = conn.cursor()
    #test mean difference - using f1, MSU & ARL
    fname = "/home/rastrom/w00_cognition/analysis_data/testplot.png"
    sql1 = "SELECT f1_score, time_order FROM results_summary_plus WHERE org = 
'ARL';" # Plus org='MSU'
    cur.execute(sql1)
    rows = cur.fetchall() # 78 values of f1, order (= 1,2,3) for MSU subjects
    order = [] # points for scatter plot - order numbers
    f1 = []   # points for scatter plot - f1 values
    f1_1 = [] # f1 values where order = 1
    f1_2 = [] # f1 values where order = 2
    f1_3 = [] # f1 values where order = 3
    for row in rows:
        n = float(row[1])
        f = float(row[0])
        order.append(n)
        f1.append(f)
        if n == 1:
            f1_1.append(f)
        elif n == 2:
            f1_2.append(f)
        elif n == 3:
            f1_3.append(f)
        else:
            print "row did not contain order no - row = "
            print row
    # End for row - Now create scatter plot
    avg_1 = numpy.mean(f1_1) # Averages, standard deviations for trial [1,2,3]
    avg_2 = numpy.mean(f1_2)
    avg_3 = numpy.mean(f1_3)
    std_1 = numpy.std(f1_1)
    std_2 = numpy.std(f1_2)
    std_3 = numpy.std(f1_3)
    n1 = len(f1_1)
    n2 = len(f1_2)
    n3 = len(f1_3)
    avgs = [avg_1,avg_2,avg_3] # For superimposed Averages plot
    t1 = functions.t_conf_int(n1, std_1, 0.95)
```

```
    t2 = functions.t_conf_int(n2, std_2, 0.95)
    t3 = functions.t_conf_int(n3, std_3, 0.95)
    orders = [1,2,3]           # For superimposed Averages plot

    # Begin plot creation
    fig = plt.figure()
    fig, ax = plt.subplots()
    # Plot 3 averages and error ranges (95%)
    ax.scatter(1.05, avg_1, s = 150, marker='.', facecolor='red') # + .05 for
easier reading on plot
    plt.errorbar(1.05, avg_1, yerr=t1, color='red')
    ax.scatter(2.05, avg_2, s = 150, marker='.', facecolor='red')
    plt.errorbar(2.05, avg_2, yerr=t2, color='red')
    ax.scatter(3.05, avg_3, s = 150, marker='.', facecolor='red')
    plt.errorbar(3.05, avg_3, yerr=t3, color='red')
    # plot line graphing 3 averages
    ax.plot(orders,avgs, color='red')
    # Scatter plot - all values versus [1,2,3]
    plt.scatter(order,f1, s = 20, marker='x', facecolor='black')
    # Labels and Axes
    plt.title("ARL - F1 Score for 1st, 2nd, 3rd Trial") # Plus "ARL ..."
    plt.xlabel("Order of trial")
    plt.xticks( [1,2,3],[ "1st", "2nd", "3rd"])
    plt.ylabel("F1 Score")
    axes = plt.gca()
    axes.set_xlim([0,4])
    axes.set_ylim([0.0,1.0])
    # Save the plot
    plt.savefig(fname)
    # Done with plot creation

    conn.close()
    print "End test analysis"
# End main
```

End order_plots.py

Appendix C. Results Summary Listing

This Appendix provides a summary listing of the results:

research=> CREATE OR REPLACE VIEW seven_scores AS SELECT org,subject,display,ROUND(CAST(completion_time AS NUMERIC),1) AS completion_time,tp,fp,tn,fn,recall,precision,f1_score FROM results_summary_plus ORDER BY subject,display;

research=> SELECT * FROM seven_scores;

```
 org | subject | display  | completion_time | tp | fp | tn | fn | recall |
precision | f1_score
-----+---------+----------+-----------------+----+----+----+----+--------+--------
---+----------
 MSU |       1 | nodelink |             4.9 | 24 | 68 | 30 | 18 |   0.57 |
0.26 |     0.36
 MSU |       1 | pc       |             1.9 | 42 | 48 | 50 |  0 |   1.00 |
0.47 |     0.64
 MSU |       1 | table    |            20.1 | 13 | 17 | 81 | 29 |   0.31 |
0.43 |     0.36
 MSU |       8 | nodelink |             4.8 | 25 | 38 | 60 | 17 |   0.60 |
0.40 |     0.48
 MSU |       8 | pc       |             4.1 | 30 | 46 | 52 | 12 |   0.71 |
0.39 |     0.51
 MSU |       8 | table    |            18.0 | 26 | 33 | 65 | 16 |   0.62 |
0.44 |     0.51
 MSU |       9 | nodelink |             5.4 | 12 | 49 | 49 | 30 |   0.29 |
0.20 |     0.23
 MSU |       9 | pc       |             7.8 | 32 | 74 | 24 | 10 |   0.76 |
0.30 |     0.43
 MSU |       9 | table    |            17.4 | 19 | 40 | 58 | 23 |   0.45 |
0.32 |     0.38
 MSU |      10 | nodelink |            10.0 | 38 |  4 | 94 |  4 |   0.90 |
0.90 |     0.90
 MSU |      10 | pc       |            11.9 |  1 | 29 | 69 | 41 |   0.02 |
0.03 |     0.03
 MSU |      10 | table    |            10.8 | 23 | 17 | 81 | 19 |   0.55 |
0.58 |     0.56
 MSU |      13 | nodelink |             4.5 | 20 | 15 | 83 | 22 |   0.48 |
0.57 |     0.52
 MSU |      13 | pc       |            13.1 |  6 | 30 | 68 | 36 |   0.14 |
0.17 |     0.15
 MSU |      13 | table    |             9.1 | 16 | 24 | 74 | 26 |   0.38 |
0.40 |     0.39
 MSU |      14 | nodelink |            19.6 | 29 | 33 | 65 | 13 |   0.69 |
0.47 |     0.56
 MSU |      14 | pc       |             5.8 | 26 | 59 | 39 | 16 |   0.62 |
0.31 |     0.41
 MSU |      14 | table    |            14.7 | 27 | 22 | 76 | 15 |   0.64 |
0.55 |     0.59
 MSU |      15 | nodelink |             2.2 | 29 | 67 | 31 | 13 |   0.69 |
0.30 |     0.42
 MSU |      15 | pc       |             2.4 | 11 | 60 | 38 | 31 |   0.26 |
0.15 |     0.19
 MSU |      15 | table    |            14.5 | 26 | 44 | 54 | 16 |   0.62 |
0.37 |     0.46
 MSU |      16 | nodelink |             5.8 | 20 | 22 | 76 | 22 |   0.48 |
0.48 |     0.48
 MSU |      16 | pc       |             9.8 | 42 | 97 |  1 |  0 |   1.00 |
0.30 |     0.46
 MSU |      16 | table    |            11.9 | 18 | 22 | 76 | 24 |   0.43 |
0.45 |     0.44
 MSU |      17 | nodelink |            11.7 | 25 | 28 | 70 | 17 |   0.60 |
0.47 |     0.53
 MSU |      17 | pc       |             4.0 | 18 |  7 | 91 | 24 |   0.43 |
0.72 |     0.54
 MSU |      17 | table    |            10.8 | 31 | 45 | 53 | 11 |   0.74 |
0.41 |     0.53
```

MSU	20	nodelink	8.1	22	26	72	20	0.52	0.46	0.49
MSU	20	pc	14.9	15	6	92	27	0.36	0.71	0.48
MSU	20	table	8.0	18	21	77	24	0.43	0.46	0.44
MSU	21	nodelink	2.9	26	47	51	16	0.62	0.36	0.45
MSU	21	pc	2.3	27	70	28	15	0.64	0.28	0.39
MSU	21	table	13.5	27	17	81	15	0.64	0.61	0.63
MSU	22	nodelink	14.6	38	9	89	4	0.90	0.81	0.85
MSU	22	pc	17.4	27	42	56	15	0.64	0.39	0.49
MSU	22	table	6.0	21	16	82	21	0.50	0.57	0.53
MSU	23	nodelink	8.7	32	19	79	10	0.76	0.63	0.69
MSU	23	pc	5.7	4	0	98	38	0.10	1.00	0.17
MSU	23	table	5.8	27	11	87	15	0.64	0.71	0.68
MSU	24	nodelink	7.9	26	44	54	16	0.62	0.37	0.46
MSU	24	pc	3.5	29	47	51	13	0.69	0.38	0.49
MSU	24	table	12.8	16	58	40	26	0.38	0.22	0.28
MSU	28	nodelink	3.6	21	57	41	21	0.50	0.27	0.35
MSU	28	pc	5.3	36	48	50	6	0.86	0.43	0.57
MSU	28	table	19.1	23	52	46	19	0.55	0.31	0.39
MSU	29	nodelink	3.1	11	5	93	31	0.26	0.69	0.38
MSU	29	pc	9.6	5	2	96	37	0.12	0.71	0.20
MSU	29	table	11.7	40	94	4	2	0.95	0.30	0.45
MSU	30	nodelink	4.4	25	26	72	17	0.60	0.49	0.54
MSU	30	pc	12.0	19	33	65	23	0.45	0.37	0.40
MSU	30	table	11.7	18	45	53	24	0.43	0.29	0.34
MSU	31	nodelink	0.8	0	24	74	42	0.00	0.00	0.00
MSU	31	pc	6.0	28	12	86	14	0.67	0.70	0.68
MSU	31	table	6.2	15	15	83	27	0.36	0.50	0.42
MSU	33	nodelink	14.2	13	0	98	29	0.31	1.00	0.47
MSU	33	pc	16.2	7	10	88	35	0.17	0.41	0.24
MSU	33	table	9.6	5	0	98	37	0.12	1.00	0.21
MSU	34	nodelink	3.2	8	0	98	34	0.19	1.00	0.32
MSU	34	pc	12.6	29	16	82	13	0.69	0.64	0.67
MSU	34	table	16.0	25	9	89	17	0.60	0.74	0.66
MSU	35	nodelink	8.7	19	21	77	23	0.45	0.48	0.46
MSU	35	pc	19.4	17	21	77	25	0.40	0.45	0.43

MSU	35	table	13.6	32	8	90	10	0.76	0.80	0.78
MSU	36	nodelink	4.8	8	26	72	34	0.19	0.24	0.21
MSU	36	pc	4.8	36	0	98	6	0.86	1.00	0.92
MSU	36	table	8.8	28	27	71	14	0.67	0.51	0.58
MSU	37	nodelink	17.1	38	25	73	4	0.90	0.60	0.72
MSU	37	table	19.3	39	27	71	3	0.93	0.59	0.72
MSU	40	nodelink	9.9	36	19	79	6	0.86	0.65	0.74
MSU	40	pc	16.6	36	35	63	6	0.86	0.51	0.64
MSU	40	table	7.4	28	9	89	14	0.67	0.76	0.71
MSU	41	pc	10.1	28	46	52	14	0.67	0.38	0.48
MSU	41	table	17.4	28	54	44	14	0.67	0.34	0.45
MSU	42	nodelink	12.0	26	43	55	16	0.62	0.38	0.47
MSU	42	pc	11.8	29	6	92	13	0.69	0.83	0.75
MSU	74	nodelink	5.9	29	44	54	13	0.69	0.40	0.50
MSU	74	pc	4.5	27	46	52	15	0.64	0.37	0.47
MSU	74	table	12.4	22	48	50	20	0.52	0.31	0.39
ARL	101	nodelink	5.8	15	25	73	27	0.36	0.38	0.37
ARL	101	pc	9.4	5	20	78	37	0.12	0.20	0.15
ARL	101	table	14.1	26	3	95	16	0.62	0.90	0.73
ARL	112	nodelink	13.8	33	12	86	9	0.79	0.73	0.76
ARL	112	pc	19.2	23	0	98	19	0.55	1.00	0.71
ARL	112	table	8.1	26	13	85	16	0.62	0.67	0.64
ARL	128	nodelink	11.8	23	12	86	19	0.55	0.66	0.60
ARL	128	pc	9.2	21	30	68	21	0.50	0.41	0.45
ARL	146	nodelink	10.6	12	9	89	30	0.29	0.57	0.38
ARL	146	table	17.3	33	2	96	9	0.79	0.94	0.86
ARL	175	nodelink	8.4	41	12	86	1	0.98	0.77	0.86
ARL	175	pc	5.8	30	19	79	12	0.71	0.61	0.66
ARL	175	table	20.3	32	22	76	10	0.76	0.59	0.67
ARL	261	nodelink	5.5	3	0	98	39	0.07	1.00	0.13
ARL	261	pc	6.7	2	27	71	40	0.05	0.07	0.06
ARL	261	table	15.5	14	44	54	28	0.33	0.24	0.28
ARL	274	nodelink	20.0	25	0	98	17	0.60	1.00	0.75
ARL	274	pc	20.0	14	0	98	28	0.33	1.00	0.50
ARL	274	table	20.0	33	3	95	9	0.79	0.92	0.85

ARL	298	nodelink	10.9	20	4	94	22	0.48	0.83	0.61
ARL	298	pc	12.2	23	20	78	19	0.55	0.53	0.54
ARL	298	table	20.0	15	3	95	27	0.36	0.83	0.50
ARL	333	nodelink	15.8	30	4	94	12	0.71	0.88	0.79
ARL	333	pc	1.6	32	55	43	10	0.76	0.37	0.50
ARL	333	table	14.7	18	30	68	24	0.43	0.38	0.40
ARL	340	nodelink	12.7	40	0	98	2	0.95	1.00	0.98
ARL	340	pc	20.0	39	0	98	3	0.93	1.00	0.96
ARL	340	table	17.7	37	0	98	5	0.88	1.00	0.94
ARL	411	nodelink	18.3	42	28	70	0	1.00	0.60	0.75
ARL	411	pc	20.0	13	14	84	29	0.31	0.48	0.38
ARL	411	table	20.0	27	5	93	15	0.64	0.84	0.73
ARL	481	nodelink	11.5	42	2	96	0	1.00	0.95	0.98
ARL	481	pc	20.0	24	5	93	18	0.57	0.83	0.68
ARL	481	table	18.0	42	26	72	0	1.00	0.62	0.76
ARL	493	nodelink	20.0	28	2	96	14	0.67	0.93	0.78
ARL	493	pc	16.2	39	2	96	3	0.93	0.95	0.94
ARL	493	table	16.5	38	3	95	4	0.90	0.93	0.92
ARL	515	nodelink	19.9	20	33	65	22	0.48	0.38	0.42
ARL	515	pc	19.9	16	59	39	26	0.38	0.21	0.27
ARL	515	table	20.0	14	9	89	28	0.33	0.61	0.43
ARL	597	nodelink	7.2	42	34	64	0	1.00	0.55	0.71
ARL	597	pc	3.5	22	97	1	20	0.52	0.18	0.27
ARL	597	table	11.5	33	28	70	9	0.79	0.54	0.64
ARL	674	nodelink	5.6	27	36	62	15	0.64	0.43	0.51
ARL	674	pc	1.5	32	55	43	10	0.76	0.37	0.50
ARL	674	table	13.4	21	25	73	21	0.50	0.46	0.48
ARL	678	nodelink	20.0	32	55	43	10	0.76	0.37	0.50
ARL	678	pc	12.7	23	34	64	19	0.55	0.40	0.46
ARL	678	table	20.0	23	52	46	19	0.55	0.31	0.39
ARL	734	table	12.6	17	3	95	25	0.40	0.85	0.55
ARL	747	nodelink	6.9	23	32	66	19	0.55	0.42	0.47
ARL	747	pc	15.1	16	24	74	26	0.38	0.40	0.39
ARL	747	table	6.4	22	23	75	20	0.52	0.49	0.51
ARL	817	nodelink	12.3	33	7	91	9	0.79	0.83	0.80

```
 ARL |      817 | pc       |            10.8 |  1 | 57 | 41 | 41 |   0.02 |
0.02 |      0.02
 ARL |      817 | table    |            16.1 | 22 | 31 | 67 | 20 |   0.52 |
0.42 |      0.46
 ARL |      840 | nodelink |             8.5 | 10 |  0 | 98 | 32 |   0.24 |
1.00 |      0.38
 ARL |      840 | pc       |            14.7 |  9 |  0 | 98 | 33 |   0.21 |
1.00 |      0.35
 ARL |      840 | table    |            11.9 | 11 |  0 | 98 | 31 |   0.26 |
1.00 |      0.42
 ARL |      874 | nodelink |            14.5 |  3 |  1 | 97 | 39 |   0.07 |
0.75 |      0.13
 ARL |      874 | table    |            20.0 | 21 |  5 | 93 | 21 |   0.50 |
0.81 |      0.62
 ARL |      913 | nodelink |            14.5 | 25 |  7 | 91 | 17 |   0.60 |
0.78 |      0.68
 ARL |      913 | pc       |             7.8 | 14 | 55 | 43 | 28 |   0.33 |
0.20 |      0.25
 ARL |      913 | table    |            20.0 | 22 |  7 | 91 | 20 |   0.52 |
0.76 |      0.62
 ARL |      921 | nodelink |             4.2 | 26 | 47 | 51 | 16 |   0.62 |
0.36 |      0.45
 ARL |      921 | pc       |             3.4 | 28 | 66 | 32 | 14 |   0.67 |
0.30 |      0.41
 ARL |      921 | table    |             3.7 | 23 | 56 | 42 | 19 |   0.55 |
0.29 |      0.38
(145 rows)
```

List of Symbols, Abbreviations, and Acronym

ARL	US Army Research Laboratory
FN	false negative
FP	false positive
FY	fiscal year
ID	identification
IDS	intrusion detection system
IPS	intrusion prevention system
MSU	Morgan State University
NSD	Network Science Division
OS	operating system
PC	parallel coordinate
SQL	structured query language
TN	true negative
TP	true positive

1 (PDF)	DEFENSE TECH INFO CTR ATTN DTIC OCA
2 (PDF)	US ARMY RSRCH LAB ATTN IMAL HRA MAIL & RECORDS MGMT ATTN RDRL CIO LL TECHL LIB
1 (PDF)	GOVT PRNTG OFC ATTN A MALHOTRA
5 (PDF)	US ARMY RSRCH LAB ATTN RDRL CIN D R ASTROM R ERBACHER W GLODEK P RITCHEY S HUTCHINSON